

Graph-Induced Multilinear Maps from Lattices

Craig Gentry
IBM

Sergey Gorbunov*
MIT

Shai Halevi
IBM

November 11, 2014

Abstract

Graded multilinear encodings have found extensive applications in cryptography ranging from non-interactive key exchange protocols, to broadcast and attribute-based encryption, and even to software obfuscation. Despite seemingly unlimited applicability, essentially only two candidate constructions are known (GGH and CLT). In this work, we describe a new graph-induced multilinear encoding scheme from lattices. In a graph-induced multilinear encoding scheme the arithmetic operations that are allowed are restricted through an explicitly defined directed graph (somewhat similar to the “asymmetric variant” of previous schemes). Our construction encodes Learning With Errors (LWE) samples in short square matrices of higher dimensions. Addition and multiplication of the encodings corresponds naturally to addition and multiplication of the LWE secrets.

*Email: sergeyg@mit.edu. Work partially performed at IBM T.J. Watson Research Center. Supported by Microsoft PhD fellowship.

Contents

1	Introduction	1
1.1	Our Results	1
1.1.1	Our Techniques	2
1.2	Applications	3
1.3	Organization	4
2	Preliminaries	5
2.1	Lattice Preliminaries	5
2.1.1	Gaussian Distributions	5
2.1.2	Trapdoors for Lattices	6
2.1.3	Leftover Hash Lemma Over Gaussians	6
2.2	Graded Multilinear Encodings	6
2.2.1	Syntax of Graph-Induced Graded Encoding Schemes	7
2.2.2	Correctness	8
2.2.3	Variations	9
3	Our Graph-Induced Multilinear Maps	9
3.1	Correctness	11
3.2	A Commutative Variant	12
3.3	Public Sampling and Some Other Variations	13
4	Cryptanalysis	13
4.1	Encoding of Zero is a Weak Trapdoor	14
4.2	Recovering Hidden $\mathbf{A}_v$'s.	15
5	Applications	17
5.1	Multipartite Key-Agreement	17
5.2	Candidate Branching-Program Obfuscation	19
5.2.1	A Concrete BP-Obfuscation Candidate	21
	Bibliography	22
A	Parameter Selection	26

1 Introduction

Cryptographic multilinear maps are an amazingly powerful tool: like homomorphic encryption schemes, they let us encode data in a manner that simultaneously hides it and permits processing on it. But they go even further and let us recover some limited information (such as equality) on the processed data without needing any secret key. Even in their simple bi-linear form (that only supports quadratic processing) they already give us pairing-based cryptography [Jou04, SOK00, BF03], enabling powerful applications such as identity- and attribute-based encryption [BF01, Wat05, GPSW06], broadcast encryption [BGW05] and many others. In their general form, cryptographic multilinear maps are so useful that we had a body of work examining their applications even before we knew of any candidate constructions to realize them [BS03, RS09, PTT10, Rot13].

Formally, a non-degenerate map between order- q algebraic groups, $e : G^d \rightarrow G_T$, is d -multilinear if for all $a_1, \dots, a_d \in \mathbb{Z}_q$ and $g \in G$,

$$e(g^{a_1}, \dots, g^{a_d}) = e(g, \dots, g)^{a_1 \cdots a_d}.$$

We say that the map e is “cryptographic” if we can evaluate it efficiently and at least the discrete-logarithm in the groups G, G_T is hard.

In a recent breakthrough, Garg, Gentry and Halevi [GGH13b] gave the first candidate construction of multilinear maps from ideal lattices, followed by a second construction by Coron, Lepoint and Tibouchi [CLT13] over the integers. (Some optimizations to the GGH scheme were proposed in [LSS14]). Due to certain differences between their construction and “ideal” multilinear maps, Garg et al. (and Coron et al.) called their constructions “graded encoding schemes.” These graded encoding schemes realize an approximate version of multilinear maps with no explicit algebraic groups, where the transformation $a \mapsto g^a$ is replaced by some (randomized) encoding function.

Moreover, these constructions are “graded”, in the sense that they allow intermediate computation. One way to think of these intermediate computations is as a sequence of levels (or groups) $G_1, \dots, G_d$ and a set of maps e_{ij} such that for all $g_i^a \in G_i, g_j^b \in G_j$ (satisfying $i + j \leq d$), $e_{ij}(g_i^a, g_j^b) = g_{i+j}^{ab}$. Asymmetric variant of graded multilinear maps provides additional structure on how these encodings can be combined. Each encoding is assigned with a set of levels $S \subseteq [N]$. Given two encodings $g_S^a, g_{S'}^b$, the map allows to compute $g_{S \cup S'}^{ab}$ only if $S \cap S' = \emptyset$.

Both [GGH13b] and [CLT13] constructions begin from some variant of homomorphic encryption and use public-key encryption as the encoding method. The main new ingredient, however, is that they also publish a defective version of the secret key, which cannot be used for decryption but can be used to test if a ciphertext encrypts a zero. (This defective key is called the “zero-test parameter”.) Over the last two years, the applications of (graded) multilinear maps have expanded much further, supporting applications such as witness encryption, general-purpose obfuscation, functional encryption, and many more [GGSW13, GGH⁺13c, GGH⁺13a, BGG⁺14, BZ14].

1.1 Our Results

We present a new “graph-induced” variant of multilinear maps. In this variant, the multilinear map is defined with respect to a directed acyclic graph. Namely, encoded value are associated with paths in the graph, and it is only possible to add encoding relative to the same paths, or to multiply encodings relative to “connected paths” (i.e., one ends where the other begins) Our candidate

construction of graph-induced multilinear maps does not rely on ideal lattices or hard-to-factor integers. Rather, *we use standard random lattices* such as those used in LWE-based cryptography. We follow a similar outline to the previous constructions, except our instance generation algorithm takes as input a description of a graph. Furthermore, our zero-tester *does not* include any secrets about the relevant lattices. Rather, in our case the zero-tester is just a random matrix, similar to a *public key* in common LWE-based cryptosystems.

Giving up the algebraic structure of ideal lattices and integers could contribute to a better understanding of the candidate itself, reducing the risk of unforeseen algebraic crypt-analytical attacks. On the flip side, using our construction is sometimes harder than previous construction, exactly because we give up some algebraic structure. For that same reason, we were not able so far to reduce any of our new construction to “nice” hardness assumptions, currently they are all just candidate constructions, that withstood our repeated cryptanalytic attempts at breaking them. Still we believe that our new construction is a well needed addition to our cryptographic toolbox, providing yet another avenue for implementing multilinear maps.

1.1.1 Our Techniques

Our starting point is the new homomorphic encryption (HE) scheme of Gentry, Sahai and Waters [GSW13]. The secret key in that scheme is a vector $\mathbf{a} \in \mathbb{Z}_q^m$, and a ciphertext encrypting $\mu \in \mathbb{Z}_q$ is a matrix $\mathbf{C} \in \mathbb{Z}_q^{m \times m}$ with small entries such that $\mathbf{C} \cdot \mathbf{a} = \mu \cdot \mathbf{a} + \mathbf{e}$ for some small error vector $\mathbf{e}$. In other words, valid ciphertexts all have the secret key $\mathbf{a}$ as an “approximate eigenvector”, and the eigenvalue is the message. Given the secret eigenvector $\mathbf{a}$, decoding arbitrary μ ’s becomes easy.

This HE scheme supports addition and multiplication, but we also need a *public* equivalent of the approximate eigenvector for zero-testing. The key idea is to replace the “approximate eigenvector” with an “approximate eigenspace” by increasing the dimensions. Instead of having a single approximate eigenvectors, our “approximate eigenspace” is described by n vectors $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$. The approximate eigenvalues will not merely be elements of $\mathbb{Z}_q$, but rather matrices $\mathbf{S} \in \mathbb{Z}_q^{n \times n}$ with small entries. An encoding of $\mathbf{S}$ is a matrix $\mathbf{C} \in \mathbb{Z}_q^{m \times m}$ with small entries such that

$$\mathbf{C} \cdot \mathbf{A} = \mathbf{A} \cdot \mathbf{S} + \mathbf{E}$$

for small noise matrix $\mathbf{E} \in \mathbb{Z}_q^{m \times n}$. In other words, $\mathbf{C}$ is a matrix that maps any column vector in $\mathbf{A}$ to a vector that is very close to the span of $\mathbf{A}$. In that sense, $\mathbf{A}$ is an approximate eigenspace. In the HE scheme, $\mathbf{a}$ was a secret key that allowed us to easily recover μ . However, for the eigenspace setting, assuming $\mathbf{A}$ is just a uniformly random matrix and $\mathbf{S}$ is a random small matrix, $\mathbf{A} \cdot \mathbf{S} + \mathbf{E}$ is an LWE instance that looks uniform even when given $\mathbf{A}$.

Overview of Our Construction. Our construction is parametrized by a directed acyclic graph $G = (V, E)$. For each node $v \in V$, we assign a random matrix $\mathbf{A}_v \in \mathbb{Z}_q^{m \times n}$. Any path $u \rightsquigarrow v$ (which can be a single edge) can be assigned with an encoding $\mathbf{D} \in \mathbb{Z}_q^{m \times m}$ of some plaintext secret $\mathbf{S} \in \mathbb{Z}_q^{n \times n}$ satisfying

$$\mathbf{D} \cdot \mathbf{A}_u = \mathbf{A}_v \cdot \mathbf{S} + \mathbf{E} \tag{1}$$

for some small error $\mathbf{E} \in (\chi)^{m \times n}$.

Adding and multiplying encodings corresponds to addition and multiplication of matrices. Addition of encodings can only be performed relative to the same path $u \rightsquigarrow v$. For example,

given encodings $\mathbf{D}_1, \mathbf{D}_2$ at path $u \rightsquigarrow v$, we have that:

$$(\mathbf{D}_1 + \mathbf{D}_2) \cdot \mathbf{A}_u \approx \mathbf{A}_v \cdot \mathbf{S}_1 + \mathbf{A}_v \cdot \mathbf{S}_2 = \mathbf{A}_v \cdot (\mathbf{S}_1 + \mathbf{S}_2).$$

Multiplication of encodings can only be performed when they form a complete path. That is, given encodings $\mathbf{D}_1$ and $\mathbf{D}_2$ relative to paths $u \rightsquigarrow v$ and $v \rightsquigarrow w$ respectively, we have:

$$\begin{aligned} \mathbf{D}_2 \cdot \mathbf{D}_1 \cdot \mathbf{A}_u &= \mathbf{D}_2 \cdot (\mathbf{A}_v \cdot \mathbf{S}_1 + \mathbf{E}_1) \\ &= (\mathbf{A}_w \cdot \mathbf{S}_2 + \mathbf{E}_2) \cdot \mathbf{S}_1 + \mathbf{D}_2 \cdot \mathbf{E}_1 = \mathbf{A}_w \cdot \mathbf{S}_2 \cdot \mathbf{S}_1 + \underbrace{\mathbf{E}_2 \cdot \mathbf{S}_1 + \mathbf{D}_2 \cdot \mathbf{E}_1}_{\mathbf{E}'} \end{aligned} \quad (2)$$

where $\mathbf{E}'$ is small since the errors and matrices $\mathbf{S}_1, \mathbf{D}_2$ have small entries. Furthermore, it is possible to compare two encodings with the same sink node. That is, given $\mathbf{D}_1$ and $\mathbf{D}_2$ relative to paths $u \rightsquigarrow v$ and $w \rightsquigarrow v$, it is sufficient to check if $\mathbf{D}_1 \cdot \mathbf{A}_u - \mathbf{D}_2 \cdot \mathbf{A}_w$ is small since if $\mathbf{S}_1 = \mathbf{S}_2$, then we have

$$\mathbf{D}_1 \cdot \mathbf{A}_u - \mathbf{D}_2 \cdot \mathbf{A}_w = (\mathbf{A}_v \cdot \mathbf{S}_1 + \mathbf{E}_1) - (\mathbf{A}_v \cdot \mathbf{S}_2 + \mathbf{E}_2) = \mathbf{E}_1 - \mathbf{E}_2 \quad (3)$$

Hence, the random matrices $\mathbf{A}_u, \mathbf{A}_w \in \mathbb{Z}_q$, which are commonly available in the public parameters, is sufficient for comparison and zero-testing.

As we explain in Section 3, generating the encoding matrices requires knowing a trapdoor for the matrices $\mathbf{A}_i$. But for the public-sampling setting, it is possible to generate encodings of many random matrices during setup, and later anyone can take a random linear combinations of them to get “fresh” random encodings.

We remark that since $\mathbf{S}$ needs to be small in Eqn. (2), our scheme only supports encoding of *small plaintext elements*, as opposed to arbitrary plaintext elements as in previous schemes.¹ Another difference is that in the basic construction our plaintext space is a non-commutative ring (i.e. square matrices). We extend to the commutative setting in Section 3.2.

Variations and parameters. We also describe some variations of the basic scheme above, aimed at improving the parameters or offering different trade-offs. One standard way of improving parameters is to switch to a ring-LWE setting, where scalars are taken from a large polynomial ring (rather than being just integers), and the dimension of vectors and matrices is reduced proportionally. In our context, we can also use the same approach to move to a commutative plaintext space, see Section 3.2.

1.2 Applications

Our new constructions support many of the known cryptographic uses of graded encoding. Here we briefly sketch two of them.

¹The only exception is that the leftmost plaintext matrix $\mathbf{S}$ in a product could encode a large element, as Eqn. (2) is not affected by the size of $\mathbf{S}_1$. Similarly the rightmost encoding matrix $\mathbf{D}$ in a product need not be small. We do not use these exceptions in the current paper, however.

Non-interactive Multipartite Key-Exchange. Consider k -partite key-exchange. We design a graph in a star topology with k -branches each of length $k - 1$ nodes. All branches meet at the common sink node $\mathbf{A}_0$. For each branch i , we associate encodings of small LWE secrets $t_1, \dots, t_k$ in a specific order. The public parameters consists of many such plaintext values t_i s and their associated encodings. Each player j takes a random linear combination of these encodings. It stores one of the encodings along the path as the secret key and broadcasts the rest of to other players. Assume some canonical ordering of the players. Each player computes the $k - 1$ product of the other players' encodings along the path with index j and its own secret encoding. This yields an encoding $\mathbf{D}$ of $\mathbf{T}^* = \prod_{i \in [k]} s_i$, satisfying

$$\mathbf{D} \cdot \mathbf{A}_{j,1} = \mathbf{A}_0 \cdot \prod_{i \in [k]} s_i + \text{noise}$$

And the players obtain the shared secret key by applying a randomness extractor on the most significant bits.

Branching-program obfuscation. Perhaps the “poster application” of cryptographic graded encodings is to obtain general-purpose obfuscation [GGH⁺13c, BR14a, BGK⁺14, PST14, GLSW14], with the crucial step being the use of graded encoding to obfuscate branching programs. These branching programs are represented as a sequence of pairs of encoded matrices, and the user just picks one matrix from each pair and then multiply them all in order.

This usage pattern of graded encoding fits very well into our graph-induced scheme since these matrices are given in a pre-arranged order. We describe a candidate obfuscation construction from our multilinear map based on a path graph. Informally, to obfuscate a length- L matrix branching program $\{\mathbf{B}_{i,b}\}$, we first perform Kilian’s randomization and then encode values $\mathbf{R}_{i-1}^{-1} \mathbf{B}_{i,0} \mathbf{R}_i$ and $\mathbf{R}_{i-1}^{-1} \mathbf{B}_{i,1} \mathbf{R}_i$ relative to the edge i . The user can then compute an encoding of a product of matrices corresponding to its input. If the product $\prod_{i \in [L]} \mathbf{B}_{i,x_{\text{vari}}} = \mathbf{I}$, then the user obtains an encoding $\mathbf{D}$ satisfying:

$$\mathbf{D} \cdot \mathbf{A}_0 = \mathbf{A}_L \cdot \mathbf{I} + \text{noise}$$

Given $\mathbf{A}_L \cdot \mathbf{I} + \text{noise}'$ in the public parameters (or its encoding), the user can then learn the result of the computation by a simple comparison. We note that our actual candidate construction is more involved as we deploy additional safeguards from the literature (See Section 5.2).

1.3 Organization

In Section 2, we provide some background and present the syntax of graph-induced multilinear maps. In Section 3, we describe our basic construction in the non-commutative variant. In Subsection 3.2 we show how to extend our basic construction to commutative variant. In Section 4, we analyze the security of our construction. In Section 5 we present applications of our construction to key-exchange and obfuscation.

Acknowledgments. We thank Zvika Brakerski for pointing out to us vulnerabilities in earlier versions of this work. We also thank Vinod Vaikuntanathan for insightful discussions.

2 Preliminaries

Notation. For any integer $q \geq 2$, we let $\mathbb{Z}_q$ denote the ring of integers modulo q and we represent $\mathbb{Z}_q$ as integers in $(-q/2, q/2]$. We let $\mathbb{Z}_q^{n \times m}$ denote the set of $n \times m$ matrices with entries in $\mathbb{Z}_q$. We use bold capital letters (e.g. $\mathbf{A}$) to denote matrices, bold lowercase letters (e.g. $\mathbf{x}$) to denote vectors.

If $\mathbf{A}_1$ is an $n \times m$ matrix and $\mathbf{A}_2$ is an $n \times m'$ matrix, then $[\mathbf{A}_1 | \mathbf{A}_2]$ denotes the $n \times (m + m')$ matrix formed by concatenating $\mathbf{A}_1$ and $\mathbf{A}_2$. Similarly, if $\mathbf{A}_1, \mathbf{A}_2$ have dimensions $n \times m$ and $\mathbf{A}_2$ is an $n' \times m$, respectively, then we denote by $(\mathbf{A}_1 / \mathbf{A}_2)$ the $(n + n') \times m$ matrix formed by putting $\mathbf{A}_1$ on top of $\mathbf{A}_2$. Similar notations apply to vectors. When doing matrix-vector multiplication we usually view vectors as column vectors.

A function $f(n)$ is *negligible* if it is $o(n^{-c})$ for all $c > 0$, and we use $\text{negl}(n)$ to denote a negligible function of n . We say that $f(n)$ is *polynomial* if it is $O(n^c)$ for some $c > 0$, and we use $\text{poly}(n)$ to denote a polynomial function of n . An event occurs with *overwhelming probability* if its probability is $1 - \text{negl}(n)$. The notation $\lfloor x \rfloor$ denotes the nearest integer to x , rounding toward 0 for half-integers.

The ℓ_∞ norm of a vector is denoted by $\|\mathbf{x}\| = \max_i |x_i|$. We identify polynomials with their representation in some standard basis (e.g., the standard coefficient representation), and the norm of a polynomial is the norm of the representation vector. The norm of a matrix, $\|\mathbf{A}\|$, is the norm of its largest column.

Extractors. An efficient (n, m, ℓ, ϵ) -strong extractor is a poly-time algorithm $\text{Extract} : \{0, 1\}^n \rightarrow \{0, 1\}^\ell$ such that for any random variable W over $\{0, 1\}^n$ with min-entropy m , it holds that the statistical distance between $(\text{Extract}_\alpha(W), \alpha)$ and (U_ℓ, α) is at most ϵ . Here, α denotes the random bits used by the extractor. Universal hash functions [CW79, WC81] can extract $\ell = m - 2 \log \frac{1}{\epsilon} + 2$ nearly random bits, as given by the leftover hash lemma [HILL99]. This will be sufficient for our applications.

2.1 Lattice Preliminaries

2.1.1 Gaussian Distributions

For a real parameter $\sigma > 0$, define the spherical Gaussian function on $\mathbb{R}^n$ with parameter σ as $\rho_\sigma(\mathbf{x}) = \exp(-\pi \|\mathbf{x}\|^n / \sigma^2)$ for all $\mathbf{x} \in \mathbb{R}^n$. This generalizes to ellipsoid Gaussians, where we replace the parameter $\sigma \in \mathbb{R}$ by the (square root of the) covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$: For a rank- n matrix $\mathbf{S} \in \mathbb{R}^{m \times n}$, the ellipsoid Gaussian function on $\mathbb{R}^n$ with parameter $\mathbf{S}$ is defined by $\rho_S(\mathbf{x}) = \exp(-\pi \mathbf{x}^T (\mathbf{S}^T \mathbf{S})^{-1} \mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^n$. The ellipsoid discrete Gaussian distribution with parameter $\mathbf{S}$ over a set $L \subset \mathbb{R}^n$ is $D_{L, \mathbf{S}}(\mathbf{x}) = \rho_S(\mathbf{x}) / \rho_S(L)$, where $\rho_S(L)$ denotes $\sum_{\mathbf{x} \in L} \rho_S(\mathbf{x})$ and serves as just a normalization factor. The same notations also apply to the spherical case, $D_{L, \sigma}(\cdot)$, and in particular $D_{\mathbb{Z}^n, r}$ denotes the n -dimensional discrete Gaussian distribution.

It follows from [MR07] that when L is a lattice and σ is large enough relative to its “smoothing parameter” (alternatively its λ_n or the Gram-Schmidt norm of one of its bases), then for every point $\mathbf{c} \in \mathbb{R}^n$ we have

$$\Pr [\|\mathbf{x} - \mathbf{c}\| > \sigma \sqrt{n} : \mathbf{x} \xleftarrow{\mathbb{R}} D_{L, \sigma, \mathbf{c}}] \leq \text{negl}(n).$$

Also under the same conditions, the probability for a random sample from $D_{\mathbb{Z}^m, \sigma}$ to be $\mathbf{0}$ is negligible.

2.1.2 Trapdoors for Lattices

Lemma 2.1 (Lattice Trapdoors [Ajt99, GPV08, MP12]). *There is an efficient randomized algorithm $\text{TrapSamp}(1^n, 1^m, q)$ that, given any integers $n \geq 1$, $q \geq 2$, and sufficiently large $m = \Omega(n \log q)$, outputs a parity check matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and some ‘trapdoor information’ τ that enables sampling small solutions to $\mathbf{rA} = \mathbf{u} \pmod{q}$.*

Specifically, there is an efficient randomized algorithm PreSample such that for large enough $s = \Omega(\sqrt{n \log q})$ and with overwhelming probability over $(\mathbf{A}, \tau) \leftarrow \text{TrapSamp}(1^n, 1^m, q)$, the following two distributions are within $\text{negl}(n)$ statistical distance:

- $\mathcal{D}_1[\mathbf{A}, \tau]$ chooses a uniform $\mathbf{u} \in \mathbb{Z}_q^n$ and uses τ to solve for $\mathbf{rA} = \mathbf{u} \pmod{q}$,

$$\mathcal{D}_1[\mathbf{A}, \tau] \stackrel{\text{def}}{=} \{(\mathbf{u}, \mathbf{r}) : \mathbf{u} \leftarrow \mathbb{Z}_q^n; \mathbf{r} \leftarrow \text{PreSample}(\mathbf{A}, \tau, \mathbf{u}, s)\}.$$

- $\mathcal{D}_2[\mathbf{A}]$ chooses a Gaussian $\mathbf{r} \leftarrow D_{\mathbb{Z}^m, s}$ and sets $\mathbf{u} := \mathbf{rA} \pmod{q}$,

$$\mathcal{D}_2[\mathbf{A}] \stackrel{\text{def}}{=} \{(\mathbf{u}, \mathbf{r}) : \mathbf{r} \leftarrow D_{\mathbb{Z}^m, s}; \mathbf{u} := \mathbf{rA} \pmod{q}\}.$$

We can extend PreSample from vectors to matrices by running it k times on k different vectors $\mathbf{u}$ and concatenating the results, hence we write $\mathbf{R} \leftarrow \text{PreSample}(\mathbf{A}, \tau, \mathbf{U}, s)$.

We also note that any small-enough full rank matrix $\mathbf{T}$ (over the integers) such that $\mathbf{TA} = 0 \pmod{q}$ can be used as the trapdoor τ above. This is relevant to our scheme because in many cases an “encoding of zero” can be turned into such a trapdoor (see Section 4).

2.1.3 Leftover Hash Lemma Over Gaussians

Recent works [AGHS13, AR13] considered the setting where the columns of a matrix $\mathbf{X} \in \mathbb{Z}^{t \times k}$ are drawn independently from a “wide enough” Gaussian distribution over a lattice $L \subset \mathbb{Z}^t$, $\mathbf{x}_i \leftarrow D_{L, S}$. Once these columns are fixed, we consider the distribution $\mathcal{D}_{\mathbf{X}, \sigma}$, induced by choosing an integer vector $\mathbf{r}$ from a discrete spherical Gaussian over $\mathbb{Z}^t$ with parameter σ and outputting $y = \mathbf{X}^T \mathbf{r}$, $\mathcal{D}_{\mathbf{X}, \sigma} := \{\mathbf{X}^T \mathbf{r} : \mathbf{r} \leftarrow D_{\mathbb{Z}^t, \sigma}\}$. It turns out that with high probability over the choice of $\mathbf{X}$, the distribution $\mathcal{D}_{\mathbf{X}, \sigma}$ is statistically close to ellipsoid Gaussian $D_{L, \sigma \mathbf{X}}$ (and moreover the singular values of $\mathbf{X}$ are of size roughly $\sigma \sqrt{t}$).

Theorem 2.2 ([AGHS13, AR13]). *For integers $k \geq 1, t = \text{poly}(k), \sigma = \Omega(\sqrt{\log(k/\epsilon)})$ and $\sigma' = \tilde{\Omega}(k\sigma\sqrt{\log(1/\epsilon)})$, we have that with probability $1 - 2^{-k}$ over the choice $\mathbf{X} \leftarrow (D_{\mathbb{Z}^k, \sigma})^t$ that the statistical distance between $\mathcal{D}_{\mathbf{X}, \sigma'}$ and $D_{\mathbb{Z}^k, \sigma' \mathbf{X}^T}$ is smaller than ϵ .*

2.2 Graded Multilinear Encodings

The notion of graded encoding scheme that we relate is similar (but not exactly identical) to the GGH notion from [GGH13b]. Very roughly, a graded encoding scheme for an algebraic “plaintext ring R ” provides methods for encoding ring elements and manipulating these encodings. Namely we can sample random plaintext elements together with their encoding, can add and multiply encoded elements, can test if a given encoding encodes zero, and can also extract a “canonical representation” of a plaintext element from an encoding of that element.

2.2.1 Syntax of Graph-Induced Graded Encoding Schemes

There are several variations of graded-encoding systems in the literature, such as public/secret encoding, with/without re-randomization, symmetric/asymmetric, etc. Below we define the syntax for our scheme, which is still somewhat different than all of the above. The main differences are that our encodings are defined relative to edges of a directed graph (as opposed to levels/sets/vectors as in previous schemes), and that we only encode “small elements” from the plaintext space. Below we provide the relevant definitions, modifying the ones from [GGH13b].

Definition 2.1 (Graph-Induced Encoding Scheme). *A graph-based graded encoding scheme with secret sampling consists of the following (polynomial-time) procedures, $\mathcal{G}_{\text{es}} = (\text{PrmGen}, \text{InstGen}, \text{Sample}, \text{Enc}, \text{add}, \text{neg}, \text{mult}, \text{ZeroTest}, \text{Extract})$:*

- $\text{PrmGen}(1^\lambda, G, \mathcal{C})$: *The parameter-generation procedure takes the security parameter λ , underlying directed graph $G = (V, E)$, and the class $\mathcal{C}$ of supported circuits. It outputs some global parameters of the system gp , which includes in particular the graph G , a specification of the plaintext ring R and also a distribution χ over R .*

For example, in our case the global parameters consists of the dimension n of matrices, the modulus q and the Gaussian parameter σ .

- $\text{InstGen}(\text{gp})$: *The randomized instance-generation procedure takes the global parameters gp , and outputs the public and secret parameters sp, pp .*
- $\text{Sample}(\text{pp})$: *The sampling procedure samples an element in the the plaintext space, according to the distribution χ .*
- $\text{Enc}(\text{sp}, p, \alpha)$: *The encoding procedure takes the secret parameters pp , a path $p = u \rightsquigarrow v$ in the graph, and an element $\alpha \in R$ from the support of the Sample procedure, and outputs an encoding u_p of α relative to p .²*
- $\text{neg}(\text{pp}, u)$, $\text{add}(\text{pp}, u, u')$, $\text{mult}(\text{pp}, u, u')$. *The arithmetic procedures are deterministic, and they all take as input the public parameters and use them to manipulate encodings.*

Negation takes an encoding of $\alpha \in R$ relative to some path $p = u \rightsquigarrow v$ and outputs encoding of $-\alpha$ relative to the same path. Addition takes u, u' that encode $\alpha, \alpha' \in R$ relative to the same path p , and outputs an encoding of $\alpha + \alpha'$ relative to p . Multiplication takes u, u' that encode $\alpha, \alpha' \in R$ relative to consecutive paths $p = u \rightsquigarrow v$ and $p' = v \rightsquigarrow w$, respectively. It outputs an encoding of $\alpha \cdot \alpha'$ relative to the combined path $u \rightsquigarrow w$.

- $\text{ZeroTest}(\text{pp}, u)$: *Zero testing is a deterministic procedure that takes the public parameters pp and an encoding u that is tagged by its path p . It outputs 1 if u is an encoding of zero and 0 if it is an of a non-zero element.*
- $\text{Extract}(\text{pp}, u)$: *The extraction procedure takes as input the public parameters pp and an encoding u that is tagged by its path p . It outputs a λ -bit string that serves as a “random canonical representation” of the underlying plaintext element α (see below).*

²See the description below for the meaning of “ u_p is an encoding of α relative to p ”, formally u_p is just a bit string, which is tagged with its path p .

2.2.2 Correctness

The graph G , in conjunction with the procedures for sampling, encoding, and arithmetic operations, and the class of supported circuits, implicitly define the set S_G of “valid encodings” and its partition into sets $S_G^{(\alpha)}$ of “valid encoding of α ”.

Namely, we consider arithmetic circuits whose wires are labeled by paths in G in a way that respects the permitted operations of the scheme (i.e., negation and addition have all the same labels, and multiplication has consecutive input paths and the output is labeled by their concatenation). Then S_G consists of all the encoding that can be generated by using the sampling/encoding procedures to sample plaintext elements and compute their encoding, then compute the operations of the scheme according to Π , and collect the encoding at the output of Π . An encoding $u \in S_G$ belongs to $S_G^{(\alpha)}$ if there exists such circuit Π and inputs for which Π outputs α when evaluated on plaintext elements. Of course, to be useful we require that the sets $S_G^{(\alpha)}$ form a partition of S_G .

We can also sub-divide each $S_G^{(\alpha)}$ into $S_p^{(\alpha)}$ for different paths p in the graph, depending on the label of the output wire of Π (but here it is not important that these sets are disjoint), and define $S_p = \bigcup_{\alpha \in R} S_p^{(\alpha)}$.

Note that the sets $S_p^{(\alpha)}$ can be empty, for example in our construction the sampling procedure only outputs “small” plaintext values α , so a “large” β would have $S_p^{(\beta)} = \emptyset$. Below we denote the set of α ’s with non-empty encoding sets (relative to path p) by $\text{SMALL}_p \stackrel{\text{def}}{=} \{\alpha \in R : S_p^{(\alpha)} \neq \emptyset\}$, and similarly $\text{SMALL}_G \stackrel{\text{def}}{=} \{\alpha \in R : S_G^{(\alpha)} \neq \emptyset\}$.

We assume for simplicity that the sets SMALL depend only on the global parameters gp and not the specific parameters sp, pp . (This assumption holds for our construction and it simplifies the syntax below.)

We can now state the correctness conditions for zero-testing and extraction. For zero-testing we require that $\text{ZeroTest}(\text{pp}, u) = 1$ for every $u \in S^{(0)}$ (with probability one), and for every $\alpha \in \text{SMALL}_G$, $\alpha \neq 0$ it holds with overwhelming probability over instance-generation that $\text{ZeroTest}(\text{pp}, u) = 0$ for every encoding $u \in S_G^{(\alpha)}$.

For extraction, we roughly require that Extract outputs the same string on all the encodings of the same α , different strings on encodings of different α ’s, and random strings on encodings of “random α ’s.” Formally, we require the following for any global parameters gp output by PrmGen :

- For any plaintext element $\alpha \in \text{SMALL}_G$ and path p in G , with overwhelming probability over the parameters $(\text{sp}, \text{pp}) \leftarrow \text{InstGen}(\text{gp})$, there exists a single value $x \in \{0, 1\}^\lambda$ such that $\text{Extract}(\text{pp}, u) = x$ holds for all $u \in S_p^{(\alpha)}$.
- For any $\alpha \neq \alpha' \in \text{SMALL}_G$ and path p in G , it holds with overwhelming probability over the parameters $(\text{sp}, \text{pp}) \leftarrow \text{InstGen}(\text{gp})$ that for any $u \in S_p^{(\alpha)}$ and $u' \in S_p^{(\alpha')}$, $\text{Extract}(\text{pp}, u) \neq \text{Extract}(\text{pp}, u')$.
- For any path p in G and distribution $\mathcal{D}$ over SMALL_p with min-entropy 3λ or more, it holds with overwhelming probability over the parameters $(\text{sp}, \text{pp}) \leftarrow \text{InstGen}(\text{gp})$ that the induced distribution $\{\text{Extract}(\text{pp}, u) : \alpha \leftarrow \mathcal{D}, u \in S_d^{(\alpha)}\}$ is nearly uniform over $\{0, 1\}^\lambda$.

In some applications these conditions can be weakened. For example we often only need them to hold for some paths in G rather than all of them (e.g., we only care about source-to-sink paths).

2.2.3 Variations

Public sampling of encoded elements. One useful variation allows a public sampling procedure that takes as input $\mathbf{pp}$ rather than $\mathbf{sp}$ and outputs both a plaintext α and its encoding u_p relative to some path p . In many cases it is easy to go from secret-encoding to public sampling. Specifically, given a scheme that supports secret encoding we can augment the instance-generation procedure by sampling many tuples (α_i, u_i) relative to relevant paths (e.g., the edges in G) and adding them to the public parameters. Then a public sampling procedure can just use a subset sum of these tuples as a new sample, which would have some other distribution χ' .

If the distribution χ of the secret sampling procedure was uniform over R , then by the leftover hash lemma so is the distribution χ' of the public sampling procedure. Similarly, if χ was a Gaussian then using the Gaussian leftover-lemma Theorem 2.2 also χ' is a Gaussian (with somewhat different parameters).

In our construction we have a Gaussian distribution χ , so we can use this method to transform our scheme to one with a public sampling procedure.

Re-randomization. In some cases one may want to re-randomize a given encoding with changing the encoded value or the path in G , or to compute given a plaintext element the corresponding encoding relative to some path. Our construction does not support re-randomization (see Section 4).

3 Our Graph-Induced Multilinear Maps

The plaintext space in our basic scheme is the non-commutative ring of matrices $R = \mathbb{Z}_q^{n \times n}$, later in Section 3.2 we describe a commutative variant. In this section we only deal with correctness of these schemes, their security is discussed in Section 4.

As sketched in the introduction, for the basic scheme we have an underlying directed acyclic graph $G = (V, E)$, we identify a random matrix $\mathbf{A}_v \in \mathbb{Z}_q^{m \times n}$ with each node $v \in V$, and encodings in the scheme are defined relative to paths. A small plaintext matrix $\mathbf{S} \in R$ is encoded wrt to the path $u \rightsquigarrow v$ via another small matrix $\mathbf{D} \in \mathbb{Z}_q^{m \times m}$ such that $\mathbf{D} \cdot \mathbf{A}_u \approx \mathbf{A}_v \cdot \mathbf{S}$. In more detail, we have the following graded encoding scheme $\mathcal{G}_{\text{es}} = (\text{PrmGen}, \text{InstGen}, \text{Sample}, \text{Enc}, \text{add}, \text{neg}, \text{mult}, \text{ZeroTest}, \text{Extract})$:

- $\text{PrmGen}(1^\lambda, G, \mathcal{C})$: On input the security parameter λ , an underlying DAG $G = (V, E)$, and class $\mathcal{C}$ of supported circuits, we compute:
 1. LWE parameters n, m, q and error distribution $\chi = D_{\mathbb{Z}, s}$.
 2. A Gaussian parameters σ for PreSample .
 3. Another parameter t for the number of most significant bits used for zero-test and extraction.

The constraints that dictate these parameters are described in Appendix A. The resulting parameters for a DAG of diameter d are $n = \Theta(d\lambda \log(d\lambda))$, $q = (d\lambda)^{\Theta(d)}$, $m = \Theta(nd \log q)$, $s = \sqrt{n}$, $\sigma = \sqrt{n(d+1) \log q}$, and $t = \lfloor (\log q)/4 \rfloor - 1$. These global parameters $\mathbf{gp}$ (including the graph G) are given to all the procedures below.

- $\text{InstGen}(\mathbf{gp})$: Given the global parameters, instance-generation proceeds as follows:

1. Use trapdoor-sampling to generate $|V|$ matrices with trapdoors, one for each node.

$$\forall v \in V, (\mathbf{A}_v, \tau_v) \leftarrow \text{TrapSamp}(1^n, 1^m, q)$$

2. Choose the randomness-extractor seed β from a pairwise-independent function family, and a uniform “shift matrix” $\Delta \in \mathbb{Z}_q^{m \times n}$.

The public parameters are $\text{pp} := (\{\mathbf{A}_v : v \in V\}, \beta, \Delta)$ and the secret parameters include also the trapdoors $\{\tau_v : v \in V\}$.

- **Sample(pp)**: This procedure just samples an LWE secret $\mathbf{S} \leftarrow (\chi)^{n \times n}$ as the plaintext.
- **Enc(sp, p, S)**: On input the matrices $\mathbf{A}_u, \mathbf{A}_v$, the trapdoor τ_u , and the small matrix $\mathbf{S}$, sample an LWE error matrix $\mathbf{E}_i \leftarrow (\chi)^{m \times n}$, set $\mathbf{V} = \mathbf{A}_v \cdot \mathbf{S} + \mathbf{E} \in \mathbb{Z}_q^{m \times n}$, and then use the trapdoor τ_u to compute the encoding $\mathbf{D}_p$ s.t. $\mathbf{D}_p \cdot \mathbf{A}_u = \mathbf{V}$, $\mathbf{D}_p \leftarrow \text{PreSample}(\mathbf{A}_u, \tau_u, \mathbf{V}, \sigma)$. The output is the plaintext S and encoding $\mathbf{D}_p$.
- The arithmetic operations are just matrix operations in $\mathbb{Z}_q^{m \times m}$:

$$\text{neg}(\text{pp}, \mathbf{D}) := -\mathbf{D}, \quad \text{add}(\text{pp}, \mathbf{D}, \mathbf{D}') := \mathbf{D} + \mathbf{D}', \quad \text{and} \quad \text{mult}(\text{pp}, \mathbf{D}, \mathbf{D}') := \mathbf{D} \cdot \mathbf{D}'.$$

To see that negation and addition maintain the right structure, let $\mathbf{D}, \mathbf{D}' \in \mathbb{Z}_q^{m \times m}$ be two encodings relative to the same path $u \rightsquigarrow v$. Namely $\mathbf{D} \cdot \mathbf{A}_u = \mathbf{A}_v \cdot \mathbf{S} + \mathbf{E}$ and $\mathbf{D}' \cdot \mathbf{A}_u = \mathbf{A}_v \cdot \mathbf{S}' + \mathbf{E}'$, with the matrices $\mathbf{D}, \mathbf{D}', \mathbf{E}, \mathbf{E}', \mathbf{S}, \mathbf{S}'$ all small. Then we have

$$\begin{aligned} -\mathbf{D} \cdot \mathbf{A}_u &= \mathbf{A}_v \cdot (-\mathbf{S}) + (-\mathbf{E}), \\ \text{and } (\mathbf{D} + \mathbf{D}') \cdot \mathbf{A}_u &= (\mathbf{A}_v \cdot \mathbf{S} + \mathbf{E}) + (\mathbf{A}_v \cdot \mathbf{S}' + \mathbf{E}') = \mathbf{A}_v \cdot (\mathbf{S} + \mathbf{S}') + (\mathbf{E} + \mathbf{E}'), \end{aligned}$$

and all the matrices $-\mathbf{D}, -\mathbf{S}, -\mathbf{E}, \mathbf{D} + \mathbf{D}', \mathbf{S} + \mathbf{S}', \mathbf{E} + \mathbf{E}'$ are still small. For multiplication, consider encodings $\mathbf{D}, \mathbf{D}'$ relative to paths $v \rightsquigarrow w$ and $u \rightsquigarrow v$, respectively, then we have

$$\begin{aligned} (\mathbf{D} \cdot \mathbf{D}') \cdot \mathbf{A}_u &= \mathbf{D} \cdot (\mathbf{A}_v \cdot \mathbf{S}' + \mathbf{E}') \\ &= (\mathbf{A}_w \cdot \mathbf{S} + \mathbf{E}) \cdot \mathbf{S}' + \mathbf{D} \cdot \mathbf{E}' = \mathbf{A}_w \cdot (\mathbf{S} \cdot \mathbf{S}') + \underbrace{(\mathbf{E} \cdot \mathbf{S}' + \mathbf{D} \cdot \mathbf{E}')}_{\mathbf{E}''}, \end{aligned}$$

and the matrices $\mathbf{D} \cdot \mathbf{D}', \mathbf{S} \cdot \mathbf{S}'$, and $\mathbf{E}''$ are still small.

Of course, the matrices $\mathbf{D}, \mathbf{S}, \mathbf{E}$ all grow with arithmetic operations, but our parameter-choice ensures that for any encoding relative to any path in the graph $u \rightsquigarrow v$ (of length $\leq d$) we have $\mathbf{D} \cdot \mathbf{A}_u = \mathbf{A}_v \cdot \mathbf{S} + \mathbf{E}$ where $\mathbf{E}$ is still small, specifically $\|\mathbf{E}\| < q^{3/4} \leq q/2^{t+1}$.

- **ZeroTest(pp, D)**. Given an encoding $\mathbf{D}$ relative to path $u \rightsquigarrow v$ and the matrix $\mathbf{A}_u$, our zero-test procedure outputs 1 if and only if $\|\mathbf{D} \cdot \mathbf{A}_u\| < q/2^{t+1}$.
- **Extract(pp, D)**: Given an encoding $\mathbf{D}$ relative to path $u \rightsquigarrow v$, the matrix $\mathbf{A}_u$ and shift-matrix Δ , and the extractor seed β , we compute $\mathbf{D} \cdot \mathbf{A}_u + \Delta$, collect the t most-significant bits from each entry (when mapped to the interval $[0, q-1]$), and apply the randomness extractor, outputting

$$w := \text{RandExt}_\beta(\text{msb}_t(\mathbf{D} \cdot \mathbf{A}_u + \Delta))$$

3.1 Correctness

Correctness of the scheme follows from our invariant, which says that encoding of some plaintext matrix $\mathbf{S}$ relative to any path $u \rightsquigarrow v$ of length $\leq d$ satisfies $\mathbf{D} \cdot \mathbf{A}_u = \mathbf{A}_v \cdot \mathbf{S} + \mathbf{E}$ for $\|\mathbf{E}\| < q/2^{t+1}$.

Correctness of Zero-Test. An encoding of zero satisfies $\mathbf{D} \cdot \mathbf{A}_u = \mathbf{E}$, hence $\|\mathbf{D} \cdot \mathbf{A}_u\| < q/2^{t+1}$. On the other hand, since $\mathbf{A}_v$ is uniform then for any nonzero $\mathbf{S}$ we only get $\|\mathbf{A}_v \cdot \mathbf{S}\| \leq q/2^t$ with exponentially small probability, and since $\|\mathbf{E}\| < q/2^{t+1}$ then

$$\|\mathbf{D} \cdot \mathbf{A}_u\| \geq \|\mathbf{A}_v \cdot \mathbf{S}\| - \|\mathbf{E}\| > q/2^t - q/2^{t+1} \geq q/2^{t+1}.$$

Hence with overwhelming probability over the choice of $\mathbf{A}_v$, our zero-test will output 0 on *all* the encoding of $\mathbf{S}$.

Correctness of Extraction. We begin by proving that for any plaintext matrix $\mathbf{S}$ and any encoding $\mathbf{D}$ of $\mathbf{S}$ (relative to $u \rightsquigarrow v$), with overwhelming probability over the parameters we have that $\text{msb}_t(\mathbf{D} \cdot \mathbf{A}_u + \Delta) = \text{msb}_t(\mathbf{A}_v \cdot \mathbf{S} + \Delta)$.

Since the two matrices $\mathbf{M} = \mathbf{A}_v \cdot \mathbf{S} + \Delta$ and $\mathbf{M}' = \mathbf{D} \cdot \mathbf{A}_u + \Delta$ differ in each entry by at most $q/2^{t+1}$ modulo q , they can only differ in their top t bits due to the mod- q reduction, i.e., if for some entry we have $[\mathbf{M}]_{k,\ell} \approx 0$ but $[\mathbf{M}']_{k,\ell} \approx q$ or the other way around. (Recall that here we reduce mod- q into the interval $[0, q-1]$.) Clearly, this only happens when $\mathbf{M} \approx \mathbf{M}' \approx 0 \pmod{q}$, in particular we need

$$- < q/2^{t+1} < [\mathbf{A}_v \mathbf{S} + \Delta]_{k,\ell} < q/2^{t+1}.$$

For any $\mathbf{S}$ and $\mathbf{A}_v$, the last condition occurs only with exponentially small probability over the choice of Δ . We conclude that if all the entries of $[\mathbf{A}_v \cdot \mathbf{S} + \Delta]$ are larger than $q/2^{t+1}$ (modulo q), which happens with overwhelming probability, then for all level- i encodings $\mathbf{D}$ of $\mathbf{S}$, the top t bits of $\mathbf{D} \cdot \mathbf{A}_u$ agree with the top t bits of $\mathbf{A}_v \cdot \mathbf{S}$. We call a plaintext matrix $\mathbf{S}$ “ v -good” if the above happens, and denote their set by GOOD_v . With this notation, the arguments above say that for any fixed $\mathbf{S}, v$, we have $\mathbf{S} \in \text{GOOD}_v$ with overwhelming probability over the instance-generation randomness.

Same input implies same extracted value. For any plaintext matrix $\mathbf{S} \in \text{GOOD}_v$, clearly all its encodings relative to $u \rightsquigarrow v$ agree on the top t bits of $\mathbf{D} \cdot \mathbf{A}_u$ (since they all agree with $\mathbf{A}_v \cdot \mathbf{S}$). Hence they all have the same extracted value.

Different inputs imply different extracted values. If $\mathbf{D}, \mathbf{D}'$ encode different plaintext matrices then $\mathbf{D} - \mathbf{D}'$ is an encoding of non-zero, hence $\|(\mathbf{D} - \mathbf{D}') \cdot \mathbf{A}_u\| \gg q/2^t$ except with negligible probability, $\mathbf{D} \cdot \mathbf{A}_u + \Delta$ and $\mathbf{D}' \cdot \mathbf{A}_u + \Delta$ must differ somewhere in their top t bits. Since we use universal hashing for our randomness extractor, then with high probability (over the hash function β) we get $\text{RandExt}_\beta(\text{msb}_t(\mathbf{D} \cdot \mathbf{A}_u + \Delta)) \neq \text{RandExt}_\beta(\text{msb}_t(\mathbf{D}' \cdot \mathbf{A}_u + \Delta))$.

Random input implies random extracted value. Fix some high-entropy distribution $\mathcal{D}$ over inputs $\mathbf{S}$. Since for every $\mathbf{S}$ we have $\Pr[\mathbf{S} \in \text{GOOD}_v] = 1 - \text{negl}(\lambda)$ then also with overwhelming probability over the parameters we have $\Pr_{\mathbf{S} \leftarrow \mathcal{D}}[\mathbf{S} \in \text{GOOD}_v] = 1 - \text{negl}(\lambda)$. It is therefore enough to show that $\text{RandExt}_\beta(\text{msb}_t(\mathbf{A}_v \cdot \mathbf{S} + \Delta))$ is nearly uniform on $\mathbf{S} \leftarrow \mathcal{D}$.

We observe that the function $H(\mathbf{S}) = \mathbf{A}_v \cdot \mathbf{S} + \mathbf{\Delta}$ is itself pairwise independent on each column of the output separately, and therefore so is the function $H'(\mathbf{S}) = \text{msb}_t(H(\mathbf{S}))$.³ We note that H' has very low collision probability, its range has many more than 6λ bits in every column, so for every $\mathbf{S} \neq \mathbf{S}'$ we get $\Pr_{H'}[H'(\mathbf{S}) = H'(\mathbf{S}')] \ll 2^{-6\lambda}$. Therefore H' is a good condenser, i.e., if the min-entropy of $\mathcal{D}$ is above 3λ , then with overwhelming probability over the choice of H , the min-entropy of $H'(\mathcal{D})$ is above $3\lambda - 1$ (say). By the extraction properties of RandExt , this implies that $\text{RandExt}_\beta(H'(\mathcal{D}))$ is close to uniform (whp over β).

3.2 A Commutative Variant

In some applications it may be convenient or even necessary to work with a commutative plaintext space. Of course, simply switching to a commutative sub-ring of the ring of matrices (such as $s \cdot I$ for a scalar s and the identity I) would be insecure, but we can make it work by moving to a larger ring.

Cyclotomic rings. We switch from working over the ring of integers to working over polynomial rings, $R = \mathbb{Z}[x]/(F(X))$ and $R_q = R/qR$ for some degree n irreducible integer polynomial $F(X) \in \mathbb{Z}[X]$ and an integer $q \in \mathbb{Z}$. Elements of this ring correspond to degree- $(n-1)$ polynomials, and hence they can be represented by n -vectors of integers in some convenient basis. The norm of a ring element is the norm of its coefficient vector, and this can be extended as usual for norm of vectors and matrices over R . Addition and multiplication are just polynomial addition and multiplication modulo $F(X)$ (and also modulo q when talking about R_q).

As usual, we need a ring where the norm of a product is not much larger than the product of the norms, and this can be achieved for example by using $F = \Phi_M(X)$, the M 'th cyclotomic polynomial (of degree $n = \phi(M)$). All the required operations and lemmas that we need (such as trapdoor and pre-image sampling etc.) can be extended also to this setting, see e.g. [LPR13].

The construction remains nearly identical, except all operations are now performed over the rings R and R_q and the dimensions are changed to match. We now have the “matrices” $\mathbf{A}_v \in R_q^{m \times 1}$ with only one column (and similarly the error matrices are $\mathbf{E} \in R_q^{m \times 1}$), and the plaintext space is R_q itself. An encoding of plaintext element $s \in R_q$ relative to path $u \rightsquigarrow v$ is a small matrix $\mathbf{D} \in R_q^{m \times m}$ such that

$$\mathbf{D} \cdot \mathbf{A}_u = \mathbf{A}_v \cdot s + \mathbf{E}$$

where $\mathbf{E}$ is some small error term. As before, we only encode small plaintext elements, i.e., the sampling procedure draws s from a Gaussian distribution with small parameter. The operations all remain the same as in the basic scheme.

We emphasize that it is the plaintext space that is commutative, not the space of encoding. Indeed, if we have $\mathbf{D}, \mathbf{D}'$ that encode s, s' relative to paths $v \rightsquigarrow w$ and $u \rightsquigarrow v$, respectively, we can only multiply them in the order $\mathbf{D} \cdot \mathbf{D}'$. Multiplying in the other order is inconsistent with the graph G and hence is unlikely to yield a meaningful result. What makes the symmetric scheme useful is the ability to multiply the *plaintext elements* in arbitrary order. For example for $\mathbf{D}, \mathbf{D}'$ that encode s, s' relative to paths $u \rightsquigarrow w$ and $v \rightsquigarrow w$, we can compute either $\mathbf{D} \cdot \mathbf{A}_u \cdot s'$ or $\mathbf{D}' \cdot \mathbf{A}_v \cdot s$ and the results will both be close $\mathbf{A}_w \cdot ss'$ (and hence also close to each other).

³If q is not a power of two then H' does not produce uniformly random t -bit strings. But still its outputs on any two $\mathbf{S}' \neq \mathbf{S}$ are independent, and each has almost full (min-)entropy, which suffices for our purposes.

3.3 Public Sampling and Some Other Variations

As mentioned in Appendix 2.2.1, we can provide a public sampling procedure relative to any desired path $p = u \rightsquigarrow v$ by publishing with the public parameters a collection of pairs generated by the secret sampling procedure above, $\{(\mathbf{S}_k, \mathbf{D}_k) : k = 1, \dots, \ell\}$ (for some large enough ℓ). The public sampling procedure then takes a random linear combination of these pairs as a new sample, namely it chooses $\mathbf{r} \leftarrow D_{\mathbb{Z}^\ell, \sigma'}$ and compute the encoding pair as:

$$(\mathbf{S}, \mathbf{D}) := \left(\sum_{i \in [\ell]} \mathbf{r}_i \mathbf{S}_i, \sum_{i \in [\ell]} \mathbf{r}_i \mathbf{D}_i \right).$$

It is easy to see that the resulting $\mathbf{D}$ encodes $\mathbf{S}$ relative to the edge e . Also by Theorem 2.2, the plaintext matrix $\mathbf{S}$ is distributed according to a Gaussian distribution whp.

We note that in most applications it is not necessary to include in the public parameters the matrices for all the nodes in the graph. Indeed we typically only need the matrices for the source nodes in the DAG in order to perform zero-testing or extraction.

Some Safeguards. Since our schemes are graph-based, and hence the order of products is known in advance, we can often provide additional safeguards using Kilian-type randomization [Kil88] “on the encoding side”. Namely, for each internal node v in the graph we choose a random invertible $m \times m$ matrix modulo q $\mathbf{R}_v$, and for the sinks and sources we set $\mathbf{R}_v = I$. Then we replace each encoding $\mathbf{C}$ relative to the path $u \rightsquigarrow v$ by the masked encoding $\mathbf{C}' := \mathbf{R}_v^{-1} \cdot \mathbf{C} \cdot \mathbf{R}_u$.

Clearly, this randomization step does not affect the product on any source-to-sink path in the graph, but the masked encodings relative to any other path no longer consist of small entries, and this makes it harder to mount the attacks from Section 4. On the down side, it now takes more bits to represent these encodings.

Other safeguards of this type includes the observations that encoding matrices relative to paths that end at a sink node need not have small entries since the size of the last matrix on a path does not contribute to the size of the final error matrix. Similarly the plaintext elements that are encoded on paths that begin at source nodes need not be small, for the same reason.

We remark that applying the safeguards from above comes with a price tag: namely the encoding matrices no longer consist of small entries, hence it takes more bits to represent them.

Finally, we observe that sometimes we do not need to give explicitly the matrices $\mathbf{A}_u$ corresponding to source nodes, and can instead “fold them” into the encoding matrices. That is, instead of providing both $\mathbf{A}$ and $\mathbf{C}$ such that $\mathbf{B} = \mathbf{D} \cdot \mathbf{A} \approx \mathbf{A}' \cdot \mathbf{S}$, we can publish only the matrix $\mathbf{B}$ and keep $\mathbf{A}, \mathbf{D}$ hidden. This essentially amounts to shortening the path by one, starting it at the matrix $\mathbf{B}$. (Of course, trying to repeat this process and further process the path will lead to exponential growth in the number of matrices that we need to publish.)

4 Cryptanalysis

Below we describe several attacks and “near attacks” on some variations of our scheme, these attacks guided our choices in designing these scheme.

4.1 Encoding of Zero is a Weak Trapdoor

The main observation in this section is that an encoding of zero relative to a path $u \rightsquigarrow v$ can sometimes be used as a weak form of trapdoor for the matrix $\mathbf{A}_u$. Recall from [GPV08] that a full-rank $m \times m$ matrix $\mathbf{T}$ with small entries satisfying $\mathbf{T}\mathbf{A} = 0 \pmod{q}$ can be used as a trapdoor for the matrix $\mathbf{A}$ as per Lemma 2.1. An encoding of zero relative the path $u \rightsquigarrow v$ is a matrix $\mathbf{C}$ such that $\mathbf{C}\mathbf{A}_u = \mathbf{E} \pmod{q}$ for a small matrix $\mathbf{E}$. This is not quite a trapdoor, but it appears close and indeed we show that it can often be used as if it was a real trapdoor.

Let us denote by $\mathbf{A}'_u = (\mathbf{A}_u/I)$ the $(m+n) \times n$ matrix whose first m rows are those of $\mathbf{A}_u$ and whose last n rows are the $n \times n$ identity matrix. Given the matrices $\mathbf{A}_u$ and $\mathbf{C}$ as above, we can compute the small matrix $\mathbf{E} = \mathbf{C}\mathbf{A}_u \pmod{q}$, then set $\mathbf{C}' = [\mathbf{C} | (-\mathbf{E})]$ to be the $m \times (m+n)$ matrix whose first m columns are the columns of $\mathbf{C}$ and whose last n columns are the negation of the columns of $\mathbf{E}$. Clearly $\mathbf{C}'$ is a small matrix satisfying $\mathbf{C}'\mathbf{A}'_u = 0 \pmod{q}$, but it is not a trapdoor yet because it has rank m rather than $m+n$.

However, assume that we have two encodings of zero, relative to two (possibly different) paths that begin at the same node u . Then we can apply the procedure above to get two such matrices $\mathbf{C}'_1$ and $\mathbf{C}'_2$, and now we have $2m$ rows that are all orthogonal to $\mathbf{A}'_u \pmod{q}$, and it is very likely that we can find $m+n$ among them that are linearly independent. This gives a full working trapdoor $\mathbf{T}'_u$ for the matrix $\mathbf{A}'_u$, what can we do with this trapdoor?

Assume now that the application gives us, in addition to the zero encodings for path that begin with u , also an encoding of a plaintext elements $\mathbf{S} \neq 0$ relative to some path that ends at u , say $w \rightsquigarrow u$. This is a matrix $\mathbf{D}$ such that $\mathbf{D}\mathbf{A}_w = \mathbf{A}_u\mathbf{S} + \mathbf{E}$, namely $\mathbf{B} = \mathbf{D}\mathbf{A}_w \pmod{q}$ is an LWE instance relative to public matrix $\mathbf{A}_u$, secret $\mathbf{S}$, and error term $\mathbf{E}$. Recalling that the plaintext $\mathbf{S}$ in our scheme must be small, it is easy to convert $\mathbf{B}$ into an LWE instance relative to matrix $\mathbf{A}'_u = (\mathbf{A}_u/I)$, for which we have a trapdoor: Simply add n zero rows at the bottom, thus getting $\mathbf{B}' = (\mathbf{B}/0)$, and we have $\mathbf{B}' = \mathbf{A}'_u\mathbf{S} + \mathbf{E}'$, with $\mathbf{E}' = (\mathbf{E}/(-\mathbf{S}))$ a small matrix.⁴ Given $\mathbf{B}'$ and $\mathbf{A}'_u$, in conjunction with the trapdoor $\mathbf{T}'_u$, we can now recover the plaintext $\mathbf{S}$.

We note that a consequence of this attack is that in our scheme it is unsafe for the application to allow computation of zero-encoding, except perhaps relative to source-nodes in the graph. As we show in Section 5, it is possible to design applications that get around this problem.

Extensions. The attacks from above can be extended even to some cases where we are not given encodings of zero. Suppose that instead we are given pairs $\{(\mathbf{C}_i, \mathbf{C}'_i)\}_i$, where the two encodings in each pair encode *the same plaintext* $\mathbf{S}_i$ relative to two paths with a common end point, $u \rightsquigarrow v$ and $u' \rightsquigarrow v$. In this case we can use the same techniques to find a “weak trapdoor” for the concatenated matrix $\mathbf{A}' = (\mathbf{A}_u/\mathbf{A}_{u'})$ of dimension $2m \times n$, using the fact that $[\mathbf{C}_i | (-\mathbf{C}'_i)] \cdot \mathbf{A}' = (\mathbf{A}_v\mathbf{S}_i + \mathbf{E}_i) - (\mathbf{A}_{v'}\mathbf{S}_i + \mathbf{E}'_i) = \mathbf{E}_i - \mathbf{E}'_i$.

If we are also given a pair $(\mathbf{D}, \mathbf{D}')$ that encodes the same element $\mathbf{S}$ relative to two paths that end at u, u' , respectively, then we can use these approximate trapdoors to find $\mathbf{S}$, since $(\mathbf{D}, \mathbf{D}')$ (together with the start points of these paths) yield an LWE instance relative to public matrix $\mathbf{A}'$ and the secret $\mathbf{S}$.

Corollary 1: No Re-randomization. A consequence of the attacks above is that in our scheme we usually cannot provide encoding-of-zero in the public parameters. Hence the re-randomization

⁴ $\mathbf{B}'$ does not have the right distribution for an LWE instance, but using the trapdoor we can solve the worst-case BDD, not just the average-case LWE, so the attack still stands.

technique by adding encodings of zero usually cannot be used in our case.

Corollary 2: No Symmetric plaintext/encoding pairs. Another consequence of the attacks above is that at least in the symmetric case it is not safe to provide many pairs (s_i, C_i) s.t. C_i is an encoding of the scalar s_i along a path $u \rightsquigarrow v$. The reason is that given two such pairs $(s_1, C_1), (s_2, C_2)$ we can compute an encoding of zero along the path $u \rightsquigarrow v$ as $s_1 C_2 - s_2 C_1$.

4.2 Recovering Hidden $\mathbf{A}_v$'s.

As we noted earlier, in many applications we only need to know the matrices $\mathbf{A}_u$ for source nodes u and there is no need to publish the matrices $\mathbf{A}_v$ for internal nodes. This raises the possibility that we might get better security by withholding the $\mathbf{A}_v$'s of internal nodes.

Trying to investigate this possibility, we show below two “near attacks” for recovering the public matrices of internal nodes from those of source nodes in the graph. The first attack applies to the commutative setting, and is able to recover an approximate version of the internal matrices (with the approximation deteriorating as we move deeper into the graph). The second attack can recover the internal matrices exactly, but it requires a full trapdoor for the matrices of the source nodes (and we were not able to extend it to work with the “approximate trapdoors” that one gets from an encoding of zero).

The conclusion from these “near attacks” is uncertain. Although it is still possible that withholding the internal-node matrices helps security, it seems prudent to examine the security of candidate applications that use our scheme in a setting where the $\mathbf{A}_v$'s are all public.

Recovering the $\mathbf{A}_v$'s in the symmetric setting. For this attack we are given a matrix $\mathbf{A}_u$, and many encodings relative to the path $u \rightsquigarrow v$, *together with the corresponding plaintext elements* (e.g., as needed for the public-encoding variant). Namely, we have $\mathbf{A}_u$, small matrices $\mathbf{C}_1, \dots, \mathbf{C}_t$ (for $t > 1$) and small ring elements $s_1, \dots, s_t$ such that $\mathbf{C}_j \cdot \mathbf{A}_u = \mathbf{A}_v \cdot s_j + \mathbf{E}_j$ holds for all j , with small $\mathbf{E}_j$'s. Our goal is to find $\mathbf{A}_v$.

We note that the matrix $\mathbf{A}_v$ and the error vectors $\mathbf{E}_j$ are only defined upto small additive factors, since adding 1 to any entry in $\mathbf{A}_v$ can be offset by subtracting the s_j 's from the corresponding entry in the $\mathbf{E}_j$'s. Hence the best we can hope for is to solve for $\mathbf{A}_v$ upto a small additive factor (resp. for the $\mathbf{E}_j$'s upto a small additive multiple of the s_j 's). Denoting $\mathbf{B}_j := \mathbf{C}_j \cdot \mathbf{A}_u = \mathbf{A}_v \cdot s_j + \mathbf{E}_j$, we compute for $j = 1, \dots, t-1$,

$$\begin{aligned} \mathbf{F}_j &:= \mathbf{B}_j \cdot s_{j+1} - \mathbf{B}_{j+1} \cdot s_j \\ &= (\mathbf{A}_v \cdot s_j + \mathbf{E}_j) \cdot s_{j+1} - (\mathbf{A}_v \cdot s_{j+1} + \mathbf{E}_{j+1}) \cdot s_j = \mathbf{E}_j \cdot s_{j+1} - \mathbf{E}_{j+1} \cdot s_j. \end{aligned}$$

This gives us a non-homogeneous linear system of equations (with the s_j 's and $\mathbf{F}_j$'s as coefficients), which we want to solve for the small solution $\mathbf{E}_j$'s. Writing this system explicitly we have

$$\begin{pmatrix} [s_2] & [-s_1] & & & \\ & [s_3] & [-s_2] & & \\ & & \ddots & \ddots & \\ & & & [s_t] & [-s_{t-1}] \end{pmatrix} \begin{pmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \\ \vdots \\ \mathbf{X}_{t-1} \\ \mathbf{X}_t \end{pmatrix} = \begin{pmatrix} \mathbf{F}_1 \\ \mathbf{F}_2 \\ \vdots \\ \mathbf{F}_{t-1} \end{pmatrix},$$

where $[s]$ denotes the $m \times m$ matrix $I_{m \times m} \cdot s$. Clearly this system is partitioned into m independent systems, each of the form

$$\begin{pmatrix} s_2 & -s_1 & & & \\ & s_3 & -s_2 & & \\ & & \ddots & \ddots & \\ & & & s_t & -s_{t-1} \end{pmatrix} \begin{pmatrix} x_{1,\ell} \\ x_{2,\ell} \\ \vdots \\ x_{t-1,\ell} \\ x_{t,\ell} \end{pmatrix} = \begin{pmatrix} f_{1,\ell} \\ f_{2,\ell} \\ \vdots \\ f_{t,\ell} \end{pmatrix},$$

with $x_{j,\ell}$, $f_{j,\ell}$ being the ℓ 'th entries of the vectors $\mathbf{X}_j, \mathbf{F}_j$, respectively. These systems are under-defined, and to get the $\mathbf{E}_i$'s we need to find *small solutions* for them. Suppressing the index ℓ , we denote these systems in matrix form by $\mathbf{M}\vec{x} = \vec{f}$, and show how to find small solutions for them.

At first glance this seems like a SIS problem so one might expect it to be hard, but here we already know a small solution for the corresponding homogeneous system, namely the solution $x_j = s_j$ for all j . Below we assume that the s_j do not all share a prime factor (i.e., that $\text{GCD}(s_1, s_2, \dots, s_t) = 1$), and also that at least one of them has a small inverse in the field of fractions of R . (These two conditions hold with good probability, see discussion in [GGH13b, Sec 4.1].)

To find a small solution for the inhomogeneous system, we begin by computing an arbitrary solution for it over the ring R (not modulo q). We note that a solution exists (in particular the E_j 's solve this system over R without mod- q reduction), and we can use Gaussian elimination in the field of fractions of R to find it. Denote that solution that was found by $\vec{g} \in R$, namely we have $\mathbf{M}\vec{g} = \vec{f}$.⁵ Since over R this is a $(t-1) \times t$ system then its solution space is one-dimensional. Hence every solution to this system (and in particular the small solution that we seek) is of the form $\vec{e} = \vec{g} + \vec{s} \cdot k$ for some $k \in R$.⁶

Choosing one index j such that the element $1/s_j$ in the field of fractions is small, we compute a candidate for the scalar k simply by rounding, $k' := -\lfloor g_j/s_j \rfloor$, where division happens in the field of fractions. We next prove that indeed the vector $\vec{e}' = \vec{g} + \vec{s} \cdot k'$ is a small vector over R . Clearly $\vec{e}' \in R^t$ since $k' \in R$ and $\vec{g}, \vec{s} \in R^t$, we next prove that it must be small by showing that “the right scalar k ” must be close to the scalar k' that we computed. First, observe that $e'_j = g_j + s_j \cdot k'$ must be small, since

$$e'_j = g_j + s_j \cdot k' = g_j - \lfloor g_j/s_j \rfloor \cdot s_j = g_j - (g_j/s_j + \epsilon_j) \cdot s_j = -\epsilon_j \cdot s_j,$$

with ϵ_j the rounding error. Since both ϵ_j and s_j are small, then so is e'_j .

Now consider the “real value” e_j , it too is small and is obtained as $g_j + s_j \cdot k$ for some $k \in R$. It follows that $e_j - e'_j = s_j \cdot (k - k')$ is small, and since we know that $1/s_j$ is also small then it follows that so is $k - k' = (e_j - e'_j)/s_j$. We thus conclude that $\vec{e}' = \vec{g} + k' \cdot \vec{s} = \vec{e} + (k - k') \cdot \vec{s}$ is also small.

Repeating the same procedure for all the m independent systems, we get a small solution $\{\mathbf{E}'_j, j = 1, \dots, t\}$ to the system $\mathbf{B}_j = \mathbf{A}_v \cdot s_j + \mathbf{E}'_j$. Subtracting the $\mathbf{E}'_j$'s from the $\mathbf{B}_j$'s and dividing by the s_j 's give us (an approximation of) $\mathbf{A}_v$.

⁵Using Gaussian elimination may yield a fractional solution $\vec{g}'$, but we can “round it” to an integral solution by solving for k' the equation $\vec{g}' + \vec{s} \cdot k' = 0 \pmod{1}$, then setting $\vec{g} = \vec{g}' + \vec{s} \cdot k'$.

⁶In general the scalar k may be fractional, but if $\text{GCD}(s_1, s_2, \dots, s_t) = 1$ then k must be integral.

Recovering the $\mathbf{A}_v$'s using trapdoors. Suppose that we are given $\mathbf{A}_u$, encodings $\mathbf{C}_j$ and the corresponding plaintext matrices $\mathbf{S}_j$, s.t. $\mathbf{B}_j := \mathbf{C}_j \cdot \mathbf{A}_u = \mathbf{A}_v \cdot \mathbf{S}_j + \mathbf{E}_j \pmod{q}$ for small errors $\mathbf{E}_j$. Suppose that in addition we are also given a *full working trapdoor* for the matrix $\mathbf{A}_v$, say, in the form of a small full-rank matrix $\mathbf{T}$ over R s.t. $\mathbf{T} \cdot \mathbf{A}_v = 0 \pmod{q}$. We can then use $\mathbf{T}$ to recover the errors $\mathbf{E}_j$ from the LWE instances $\mathbf{B}_j$, which can be done without knowing $\mathbf{A}_v$: Let $\mathbf{T}^{-1}$ be the inverse of $\mathbf{T}$ over R , we compute $\mathbf{E}_j \leftarrow \mathbf{T}^{-1} \cdot (\mathbf{T} \cdot \mathbf{B}_j \pmod{q})$. Once we have the error matrices $\mathbf{E}_j$ we can subtract them and get the set of equations $\mathbf{B}_j - \mathbf{E}_j = \mathbf{A}_v \cdot \mathbf{S}_j \pmod{q}$, where the entries of $\mathbf{A}_v$ are the unknowns. With sufficiently many of these equations, we can then solve for $\mathbf{A}_v$.

We note that so far we were unable to extend this attack to using the “weak trapdoor” that one gets from an encoding of zero wrt paths of the form $v \rightsquigarrow w$. Indeed the procedure from Section 4.1 for recovering a stronger trapdoor from the weak one relies on knowing $\mathbf{A}_v$.

5 Applications

5.1 Multipartite Key-Agreement

For our first application, we describe a candidate construction for a non-interactive multipartite key-agreement protocol using the commutative variant of our graph-based encoding scheme. As is usual with multipartite key-agreement from multilinear maps, each party i is contributing an encoding of some secret s_i and the shared secret is derived from an encoding of the product $s = \prod_i s_i$. However in our case we need to use extra caution to protect against the “weak trapdoor attacks” from Section 4.1.

To that end, we design our graph to ensure that the adversary is never given encodings of the same element on two paths with a common end-point, and also is not given an encoding and the corresponding plaintext on any edge. For an k -partite protocol we use a graph topology of k directed chains that meet at a common point, where the contribution of any given party appears at different edges on different chains (i.e. the first edge on one chain, the second edge on another, the third edge on a third chain, etc.)

That is, each player i has a directed path of matrices, $\mathbf{A}_{i,1}, \dots, \mathbf{A}_{i,k+1}$, all sharing the same end-point, i.e., $\mathbf{A}_{i,k+1} = \mathbf{A}_0$ for all i . Note that every chain has k edges, and for the chain “belonging” to party i we will broadcast on its edges encodings of all the secrets $s_j, j \neq i$, but not an encoding of s_i , that last encoding will only be known to party i . Party i will multiply these encodings (the one that only it knows, and all the ones that are publicly available) to get an encoding of $\prod_{j \neq i} s_j$ relative to the path $\mathbf{A}_{i,1} \rightsquigarrow \mathbf{A}_0$. Namely, a matrix $\mathbf{D}_i$ such that $\mathbf{D}_i \cdot \mathbf{A}_{i,1} \approx \mathbf{A}_0 \cdot \prod_{j \neq i} s_j$. The shared secret is then obtained by applying the extraction procedure to this $\mathbf{D}_i$.

The assignment of which secret is encoded on what edge of what chain is done in a “round robin” fashion. Specifically, the i 'th secret s_i is encoded on the j 'th edge of the chain belonging to party $i' = j - i + 1$. In other words, the secret that we encode on the edge $\mathbf{A}_{i,j} \rightarrow \mathbf{A}_{i,j+1}$ in the graph is s_{j-i+1} , with index arithmetic modulo k . An example of the assignment of secrets to edges for a 4-partite protocol is depicted in Figure 1.

Of course, we must publish encodings that would allow the parties to choose their secrets and provide encodings for them. This means that together with the public parameters we also publish encodings of many plaintext elements $\{t_{i,\ell} : i = 1, \dots, k, \ell = 1, \dots, N\}$ (for a sufficiently large N), for each $t_{i,\ell}$ we publish encoding of it relative to all the edges $\mathbf{A}_{i',i+i'-1} \rightarrow \mathbf{A}_{i',i+i'}$ for all i, i' (index arithmetic modulo $k+1$). Party i then chooses random small coefficients $r_{i,\ell}$ and computes its encoding relative to each edge $\mathbf{A}_{i',i+i'-1} \rightarrow \mathbf{A}_{i',i+i'}$ as the linear combination

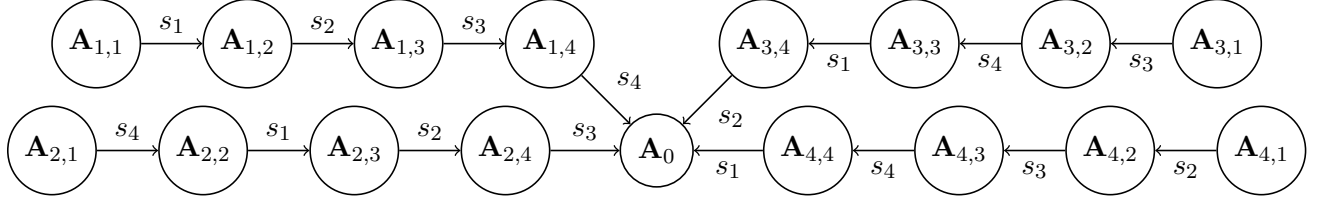


Figure 1: Graph for a 4-partite key-agreement protocol.

of the encodings on that edge with the coefficient $r_{i,\ell}$. We are now ready to describe our scheme $\mathcal{NMC}\mathcal{E} = (\text{KE.Setup}, \text{KE.Publish}, \text{KE.Keygen})$.

- $\text{KE.Setup}(1^\lambda, k)$: The setup algorithm takes as input the security parameter 1^λ and the total number of players k .
 1. Run the parameter-generation and instance-generation of our graph-based encoding scheme for the graph consisting of k chains with a common end-point, each of length k edges. Let $e_{i,j}$ denote the j 'th edge on the i 'th chain.
 2. Using the secret parameters, run the sampling procedure of the encoding scheme to choose random plaintext elements $t_{i,\ell}$ for $i = 1, \dots, k$ and $\ell = 1, \dots, N$, and for each $t_{i,\ell}$ compute also an encoding of it relative to all the edges $e_{i',j}$ for $j = i + i' \pmod k$. Denote the encoding of $t_{i,\ell}$ on chain i' (relative to edge $e_{i',i+i' \pmod k}$) by $\mathbf{C}_{i,\ell,i'}$.

The public parameters of the key-agreement protocol include the public parameters of the encoding scheme (i.e., the matrices for all the source nodes $\mathbf{A}_{i,1}$), and also the encoding matrices $\{\mathbf{C}_{i,\ell,i'} : i, i' = 1, \dots, k, \ell = 1, \dots, N\}$.

- $\text{KE.Publish}(\text{pp}, i)$: The i 'th party chooses random small plaintext elements $r_{i,\ell} \leftarrow \chi$ for $\ell = 1, \dots, N$ and then sets $\mathbf{D}_{i,i'} \leftarrow \sum_{\ell} \mathbf{C}_{i,\ell,i'} \cdot r_{i,\ell}$ for all i' . It keeps $\mathbf{D}_{i,i}$ as its secret and broadcast all the other $\mathbf{D}_{i,i'}$'s.
- $\text{KE.Keygen}(\text{pp}, i, \text{sk}_i, \{\text{pub}_j\}_{j \neq i})$: Party i collects all the matrices $\mathbf{D}_{i',i}$ (encoding the secrets $s_{i'}$ relative to “its chain” i) and orders them according to $j = i + i'$. Namely, it sets $\mathbf{F}_{j,i} = \mathbf{D}_{i+j \pmod k, i}$ for $j = 1, \dots, k$, then computes the product $\mathbf{F}_i^* = (\prod_{j=1}^k \mathbf{F}_{j,i}) \cdot \mathbf{A}_{i,1}$. Finally, party i applies the extraction procedure of the encoding scheme to obtain the secret key, setting $\text{ssk} = \text{Extract}(\mathbf{F}_i^*)$.

Security. Unfortunately, we were not able to reduce the security of this candidate scheme to any “nicer” assumption. As such, at present the only evidence of security that we can offer is the failure of our attempts to cryptanalyze it.

The basic attack from Section 4.1 does not seem to apply here since the public parameters do not provide any encoding of zero (not even relative to $\mathbf{A}_0$). Similarly, the extended attacks do not seem to apply since the only common end-point in the graph is $\mathbf{A}_0$, and no two paths that end at $\mathbf{A}_0$ include an encoding of the same element.

We note that the attacker can use the public parameters to compute an approximate trapdoors for concatenated matrices of the form $(\mathbf{A}_0 \cdot t_{i,\ell,i'} / (-\mathbf{A}_0))$ (or similar), but the broadcast messages of the parties do not provide LWE instances relative to these matrices.

Finally, we note that as for any other application of this encoding scheme, it seems that security would be enhanced by applying the additional safeguards that were discussed at the end of Section 3. That is, we can use Kilian-style randomization on the encoding side, by choosing k invertible matrices for each chain, $\mathbf{R}_{i,1}, \dots, \mathbf{R}_{i,k}$, where the first and last are set to the identity and the others are chosen at random. Then we can replacing each encoding matrix $\mathbf{C}$ in the public parameters by $\mathbf{C}' := \mathbf{R}^{-1} \cdot \mathbf{C} \cdot \mathbf{R}'$ using the randomizer matrices $\mathbf{R}, \mathbf{R}'$ belonging to the two adjacent nodes. We can also choose the first encoding matrix in each chain to have large entries.

This has no effect on the product of all the encoding matrices along the i' -th chain, but the new matrices $\mathbf{C}'$ no longer have small entries, which seems to aid security. On the down side, this increases the size of the encodings roughly by a $\log q / \log n$ factor.

5.2 Candidate Branching-Program Obfuscation

We next describe how to adapt the branching-program obfuscation constructions from previous work [GGH⁺13a, BR14b, BGK⁺14, PST14] to use our encoding schemes. We remark that on some level this is the simplest type of constructions to adapt to our setting, since we essentially need only a single chain and there almost no issues of providing zero-encoding in the public parameters (or encodings of the same plaintext relative to different nodes in the graph).

Roughly speaking, previous works all followed a similar strategy for obfuscating branching programs. Starting from a given oblivious branching program, encoded as permutation matrices, they all applied Kilian’s randomization strategy to randomized these matrices, then added some extra randomization steps (mostly multiplication by random scalars) to protect against partial-evaluation and mixed-input attacks, and finally encoded the resulting matrices relative to specially-designed sets/levels. The specific details of the extra randomization steps are somewhat different between the previous schemes, but all these techniques have their counterparts in our setting. Below we explain how to adapt the randomization techniques from previous work to our setting, and then describe one specific BP-obfuscation candidate.

Matrices vs. individual elements. Our scheme natively encodes matrices, rather than individual elements. This has some advantages, for example we need not worry about attacks that mix and match encoded elements from different matrices. At the same time it also poses some challenges, in particular some of the prior schemes worked by comparing to zero one element of the resulting matrix at the end of evaluation, an operation which is not available in our case.

To be able to examine sub-matrices (or single elements), we adopt the “bookend encoding” trick from [GGH⁺13a]. That is, we add to our chain a new source u^* and a new sink v^* , with edges from u^* to the old source and from the old sink to v^* . On the edge from u^* we encode a matrix $\mathbf{T}$ which is only nonzero in the columns that we want to examine, and on the edge to v^* we encode a matrix $\mathbf{S}$ which is only nonzero in the rows that we wish to examine. This way, we should have the matrix $\mathbf{T} \cdot \mathbf{U} \cdot \mathbf{S}$ encoded relative to a path $u^* \rightsquigarrow v^*$, and that matrix is only nonzero in the sub-matrix of interest. In the candidate below we somewhat improve on this by folding the source matrix $\mathbf{A}_{u^*}$ into the encoding of $\mathbf{T}$, publishing instead the matrix $\mathbf{A}_{u^*} \cdot \mathbf{T}$ (and in fact making $\mathbf{T}$ a single column vector).

Only small plaintexts. In our scheme we can only encode “small plaintext elements”, not every plaintext element. This is particularly relevant for Kilian randomization technique, since it requires

that we multiply by both $\mathbf{R}$ and $\mathbf{R}^{-1}$ for each randomizer matrix $\mathbf{R}$. One way to get randomizer matrices with both $\mathbf{R}$ and $\mathbf{R}^{-1}$ small is using the facts that

$$\begin{aligned} \left(\begin{array}{c|c} \mathbf{I} & \mathbf{0} \\ \hline \mathbf{R} & \mathbf{I} \end{array} \right)^{-1} &= \left(\begin{array}{c|c} \mathbf{I} & \mathbf{0} \\ \hline -\mathbf{R} & \mathbf{I} \end{array} \right), & \left(\begin{array}{c|c} \mathbf{I} & \mathbf{R} \\ \hline \mathbf{0} & \mathbf{I} \end{array} \right)^{-1} &= \left(\begin{array}{c|c} \mathbf{I} & -\mathbf{R} \\ \hline \mathbf{0} & \mathbf{I} \end{array} \right) \\ \left(\begin{array}{c|c} \mathbf{0} & \mathbf{I} \\ \hline \mathbf{I} & \mathbf{R} \end{array} \right)^{-1} &= \left(\begin{array}{c|c} \mathbf{0} & \mathbf{I} \\ \hline \mathbf{I} & -\mathbf{R} \end{array} \right), & \left(\begin{array}{c|c} \mathbf{R} & \mathbf{I} \\ \hline \mathbf{I} & \mathbf{0} \end{array} \right)^{-1} &= \left(\begin{array}{c|c} -\mathbf{R} & \mathbf{I} \\ \hline \mathbf{I} & \mathbf{0} \end{array} \right). \end{aligned}$$

Multiplying a sequence of these types of matrices above yields a high-entropy distribution of randomizer matrices with the desired property, and seemingly without obvious algebraic structure. Another family of matrices where both the matrix and its inverse are small are permutation matrices (and of course we can mix and match these families). Concretely, we speculate that a randomizer of the form

$$\mathbf{R} = \Pi_1 \cdot \begin{pmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{I} & \mathbf{R}_1 \end{pmatrix} \cdot \Pi_2 \cdot \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{R}_2 & \mathbf{I} \end{pmatrix} \cdot \Pi_3 \cdot \begin{pmatrix} \mathbf{R}_3 & \mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{pmatrix} \cdot \Pi_4 \cdot \begin{pmatrix} \mathbf{I} & \mathbf{R}_4 \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \cdot \Pi_5 \quad (4)$$

(with the Π_i 's random permutations and the $\mathbf{R}_i$'s random small matrices) has sufficient entropy and lack of algebraic structure to server as randomizers for our scheme.

We note that although these randomizers are far from uniform, there may still be hope of using some of the tools developed in [BR14b, BGK⁺14, PST14] (where the analysis includes a reduction to Kilian's information-theoretic argument). This is because the matrices before randomization are permutation matrices, and hence the random permutations Π_i can be used to perfectly randomize them. In this way, one can view the $\mathbf{R}_i$'s are merely "safeguards" to protect against possible weaknesses in the encoding scheme, and the Π_i 's are "ideal model randomizers" than can be used in an ideal-model analysis. So far we did not attempt such analysis, however.

Another way to introduce Kilian-type rerandomization in our setting is the aforementioned option of applying it "on the encoding side," i.e., choosing random $m \times m$ invertible matrices $\mathbf{P}$ modulo q and set $\mathbf{C}' \leftarrow \mathbf{P}^{-1} \cdot \mathbf{C} \cdot \mathbf{P}$.

Multiplicative binding and sraddling sets. Another difference between our setting and that of GGH or CLT is that the previous schemes support encoding relative to arbitrary subsets of a universe set, so there are exponentially many potential sets to use. In our scheme the encoding is relative to edges of a given graph, and there can be only polynomial many of them. This difference seems particularly critical in the design of sraddling sets [BGK⁺14, PST14].

On a second look, however, this issue is more a question of modeling, rather than a real difference. The different encoding sets in the "asymmetric variants" of [GGH13b, CLT13] are obtained just by multiplying by different random secret constants (e.g., the z_i 's from GGH), and we can similarly multiply our encoding matrices by such random constants mod q (especially when working over a large polynomial ring). We use that option in the candidate scheme that we describe below.

We note that similar effects can be obtained by the multiplicative binding technique of [GGH⁺13a]. Roughly speaking, the main difference between multiplicative binding and sraddling sets is that the former multiplies by constants "on the plaintext side" while the latter multiplies "on the encoding side." In our setting we can do both, and indeed it seems prudent to do so.

5.2.1 A Concrete BP-Obfuscation Candidate

For our concrete candidate below we work over a large polynomial ring of dimension k , and we will use small-dimension matrices over this ring (roughly as high as the dimension of the underlying branching program).

Let $\text{Sym}(w)$ be the set of $w \times w$ permutation matrices and consider a length- n branching program over ℓ bit inputs:

$$\text{BP} = \{(\text{ind}(i), \mathbf{B}_{i,0}, \mathbf{B}_{i,1} : i \in [n], \text{ind}(i) \in [\ell], \mathbf{B}_{i,b} \in \text{Sym}(w))\}$$

For a bit position $j \in [\ell]$, let I_j be the steps in the branching program that examines j 'th input bit: $I_j = \{i \in [n] : \text{ind}(i) = j\}$. We obfuscate BP as follows:

- Following the original construction of [GGH⁺13c] we embed the $\mathbf{B}_{i,\sigma}$'s inside higher-dimension matrices with random elements on the diagonal, but in our case it is sufficient to have only two such random entries (so the dimension only grows from w to $w + 2$). Denote the higher-dimension matrices by $\mathbf{B}'_{i,\sigma}$.

We also follow the original construction of [GGH⁺13c] by applying the same transformation to a “dummy program” DP of the same length that consists of only the identity matrices, let $\mathbf{D}'_{i,\sigma}$ be the higher-dimension dummy matrices.

- We proceed to randomize these branching programs a-la-Kilian “on the plaintext side,” by choosing randomizing matrices $\mathbf{R}_i$'s as per the form of Eqn. (4) such that both $\mathbf{R}_i$ and $\mathbf{R}_i^{-1}$ are small, and setting $\mathbf{B}''_{i,\sigma} = \mathbf{R}_{i-1} \mathbf{B}'_{i,\sigma} \mathbf{R}_i^{-1}$. The dummy program is randomized similarly.
- We then prepare $(w + 2) \times (w + 2)$ “bookend matrices” $\mathbf{S}, \mathbf{S}'$, and “bookend column vectors” $\mathbf{t}, \mathbf{t}'$. $\mathbf{S}$ is random and small except the first row which is zero, $\mathbf{t}$ is random and small except the second entry which is zero, and similarly for $\mathbf{S}'$ and $\mathbf{t}'$, subject to $\mathbf{S}' \cdot \mathbf{t}' = \mathbf{S} \cdot \mathbf{t}$. Then we set $\tilde{\mathbf{S}} = \mathbf{S} \mathbf{R}_0^{-1}$ and $\tilde{\mathbf{t}} = \mathbf{R}_n \mathbf{t}$, and similarly $\tilde{\mathbf{S}}' = \mathbf{S}' \mathbf{R}_0^{-1}$ and $\tilde{\mathbf{t}}' = \mathbf{R}_n \mathbf{t}'$.
- We also sample random small scalars $\{\alpha_{i,0}, \alpha_{i,1}, \alpha'_{i,0}, \alpha'_{i,1} : i \in [n]\}$, subject to constraint: $\prod_{i \in I_j} \alpha_{i,0} = \prod_{i \in I_j} \alpha'_{i,0}$ and $\prod_{i \in I_j} \alpha_{i,1} = \prod_{i \in I_j} \alpha'_{i,1}$. These are used for the “plaintext-side” multiplicative bundling.

We set $\mathbf{B}^*_{i,\sigma} = \mathbf{B}''_{i,\sigma} \cdot \alpha_{i,\sigma}$ for the main program and similarly $\mathbf{D}^*_{i,\sigma} = \mathbf{D}''_{i,\sigma} \cdot \alpha'_{i,\sigma}$ for the dummy program.

- Next we use our encoding scheme to encode these matrices relative to a graph with two chains with a common end-point, each of length $n + 2$. Namely we have $\mathbf{A}_1 \rightarrow \dots \rightarrow \mathbf{A}_{n+2}$ and $\mathbf{A}'_1 \rightarrow \dots \rightarrow \mathbf{A}'_{n+1} \rightarrow \mathbf{A}_{n+2}$.

For each $i \in [n]$, we encode the two matrices $\mathbf{B}^*_{n-i+1,b}$ relative to the edge $\mathbf{A}_i \rightarrow \mathbf{A}_{i+1}$, i.e., we have

$$\mathbf{C}_{n-i+1,b} \cdot \mathbf{A}_i = \mathbf{A}_{i+1} \cdot \mathbf{B}^*_{n-i+1,b} + \mathbf{E}_{i,b}$$

for some small error $\mathbf{E}_{i,b}$. Similarly we encode the dummy program with the two matrices $\mathbf{D}^*_{n-i+1,b}$ encoded relative to the edge $\mathbf{A}'_i \rightarrow \mathbf{A}'_{i+1}$, i.e.,

$$\mathbf{C}'_{n-i+1,b} \cdot \mathbf{A}'_i = \mathbf{A}'_{i+1} \cdot \mathbf{D}^*_{n-i+1,b} + \mathbf{E}'_{i,b}$$

- Encode $\tilde{\mathbf{S}}, \tilde{\mathbf{S}}'$ relative to the edges leading to the common sink, i.e. compute the encoding matrices $\mathbf{C}_S, \mathbf{C}'_{S'}$ such that

$$\mathbf{C}_S \cdot \mathbf{A}_{n+1} = \mathbf{A}_{n+2} \cdot \tilde{\mathbf{S}} + \mathbf{E}_S \text{ and } \mathbf{C}'_{S'} \cdot \mathbf{A}'_{n+1} = \mathbf{A}_{n+2} \cdot \tilde{\mathbf{S}}' + \mathbf{E}'_{S'}$$

- Compute the encoded bookend vectors, folded into the two sources $\mathbf{A}_1$ and $\mathbf{A}'_1$, namely $\mathbf{a} = \mathbf{A}_1 \cdot \tilde{\mathbf{t}} + \mathbf{e}_t$ and $\mathbf{a}' = \mathbf{A}'_1 \cdot \tilde{\mathbf{t}}' + \mathbf{e}'_t$.
- We next apply both the multiplicative bundling and the the Kilian-style randomization *also on the encoding side*. Namely we sample random full-rank matrices $\mathbf{P}_0, \dots, \mathbf{P}_n$ and $\mathbf{P}'_0, \dots, \mathbf{P}'_n$, and also random scalars modulo q $\{\beta_{i,0}, \beta_{i,1}, \beta'_{i,0}, \beta'_{i,1} : i \in [n]\}$, subject to constraints $\prod_{i \in I_j} \beta_{i,0} = \prod_{i \in I_j} \beta'_{i,0} = \prod_{i \in I_j} \beta_{i,1} = \prod_{i \in I_j} \beta'_{i,1} = 1$.

We then set $\hat{\mathbf{C}}_{i,\sigma} = \mathbf{P}_{i-1}^{-1} \cdot \mathbf{C}_{i,\sigma} \cdot \mathbf{P}_i \cdot \beta_{i,\sigma}$ and $\hat{\mathbf{C}}'_{i,\sigma} = \mathbf{P}'_{i-1}^{-1} \cdot \mathbf{C}'_{i,\sigma} \cdot \mathbf{P}'_i \cdot \beta'_{i,\sigma}$, and also $\hat{\mathbf{C}}_S = \mathbf{C}_S \cdot \mathbf{P}_0$ and $\hat{\mathbf{C}}'_{S'} = \mathbf{C}'_{S'} \cdot \mathbf{P}'_0$ and $\hat{\mathbf{a}} = \mathbf{P}_n^{-1} \mathbf{a}$ and $\hat{\mathbf{a}}' = \mathbf{P}'_n^{-1} \mathbf{a}'$.

- The obfuscation consists of all the matrices and vectors above, namely

$$\mathcal{O}(\text{BP}) = \left(\left\{ \hat{\mathbf{C}}_S, \{\hat{\mathbf{C}}_{i,\sigma} : i \in [n], \sigma \in \{0,1\}\}, \hat{\mathbf{a}} \right\}, \left\{ \hat{\mathbf{C}}'_{S'}, \{\hat{\mathbf{C}}'_{i,\sigma} : i \in [n], \sigma \in \{0,1\}\}, \hat{\mathbf{a}}' \right\} \right)$$

Evaluation. On input $\mathbf{x} \in \{0,1\}^\ell$ the user choose the appropriate encoding matrices $\hat{\mathbf{C}}_{i,0}$ or $\hat{\mathbf{C}}_{i,1}$ depending on the relevant input bit (and the same for $\hat{\mathbf{C}}'_{i,0}$ or $\hat{\mathbf{C}}'_{i,1}$) and then multiply in order setting

$$\mathbf{y} = \hat{\mathbf{C}}_S \cdot \left(\prod_{i=1}^n \hat{\mathbf{C}}_{i,x[\text{ind}(i)]} \right) \cdot \mathbf{a} = \mathbf{A}_{n+2} \cdot (\mathbf{S} \cdot \left(\prod_{i=1}^n \mathbf{B}''_{i,x[\text{ind}(i)]} \right) \cdot \mathbf{t}) + \mathbf{e}$$

and

$$\mathbf{y}' = \hat{\mathbf{C}}'_{S'} \cdot \left(\prod_{i=1}^n \hat{\mathbf{C}}'_{i,x[\text{ind}(i)]} \right) \cdot \mathbf{a}' = \mathbf{A}_{n+2} \cdot (\mathbf{S}' \cdot \left(\prod_{i=1}^n \mathbf{D}''_{i,x[\text{ind}(i)]} \right) \cdot \mathbf{t}') + \mathbf{e}'$$

The output is 1 if $\|\mathbf{y} - \mathbf{y}'\| < q^{3/4}$ and 0 otherwise. Note that indeed if $\prod_{i=1}^n \mathbf{D}_{i,x[\text{ind}(i)]} = \mathbf{I}$ then both $\mathbf{y}$ and $\mathbf{y}'$ are roughly equal to $\mathbf{A}_{n+2} \cdot \mathbf{S} \cdot \mathbf{t} \cdot (\prod_{i=1}^n \alpha_{i,x[\text{ind}(i)]})$, as needed.

Security. As before, this is merely a candidate and we do not know how to reduce its security to any “nice” assumption. However the type of attacks that we know against these scheme do not seem to apply to this candidate.

References

- [AGHS13] Shweta Agrawal, Craig Gentry, Shai Halevi, and Amit Sahai. Discrete gaussian leftover hash lemma over infinite domains. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, volume 8269 of *Lecture Notes in Computer Science*, pages 97–116. Springer Berlin Heidelberg, 2013.

- [Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *ICALP*, pages 1–9, 1999.
- [AR13] Divesh Aggarwal and Oded Regev. A note on discrete gaussian combinations of lattice vectors. *CoRR*, abs/1308.2405, 2013.
- [BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, pages 213–229, 2001.
- [BF03] Dan Boneh and Matt Franklin. Identity-based encryption from the Weil pairing. *SIAM J. of Computing*, 32(3):586–615, 2003. extended abstract in Crypto’01.
- [BGG⁺14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit abe and compact garbled circuits. In PhongQ. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556. Springer Berlin Heidelberg, 2014.
- [BGK⁺14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In PhongQ. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 221–238. Springer Berlin Heidelberg, 2014.
- [BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In Victor Shoup, editor, *Advances in Cryptology CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 258–275. Springer Berlin Heidelberg, 2005.
- [BR14a] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In Yehuda Lindell, editor, *TCC*, volume 8349 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2014.
- [BR14b] Zvika Brakerski and GuyN. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In Yehuda Lindell, editor, *Theory of Cryptography*, volume 8349 of *Lecture Notes in Computer Science*, pages 1–25. Springer Berlin Heidelberg, 2014.
- [BS03] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324:71–90, 2003.
- [BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan Garay and Rosario Gennaro, editors, *Advances in Cryptology CRYPTO 2014*, volume 8616 of *Lecture Notes in Computer Science*, pages 480–499. Springer Berlin Heidelberg, 2014.
- [CLT13] Jean-Sbastien Coron, Tancrde Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and JuanA. Garay, editors, *Advances in Cryptology CRYPTO 2013*, volume 8042 of *Lecture Notes in Computer Science*, pages 476–493. Springer Berlin Heidelberg, 2013.

- [CW79] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143 – 154, 1979.
- [GGH⁺13a] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 40–49, Oct 2013.
- [GGH13b] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and PhongQ. Nguyen, editors, *Advances in Cryptology EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, 2013. Full version at <http://eprint.iacr.org/2013/451>.
- [GGH⁺13c] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. Cryptology ePrint Archive, Report 2013/128, 2013.
- [GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC ’13, pages 467–476, New York, NY, USA, 2013. ACM.
- [GLSW14] Craig Gentry, Allison Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. Cryptology ePrint Archive, Report 2014/309, 2014. <http://eprint.iacr.org/>.
- [GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, pages 89–98, 2006.
- [GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, STOC’08*, pages 197–206. ACM, 2008.
- [GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and JuanA. Garay, editors, *Advances in Cryptology CRYPTO 2013*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer Berlin Heidelberg, 2013.
- [HILL99] Johan Hastad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28:1364–1396, March 1999.
- [Jou04] Antoine Joux. A one round protocol for tripartite diffiehellman. *Journal of Cryptology*, 17(4):263–276, 2004.
- [Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC ’88, pages 20–31, New York, NY, USA, 1988. ACM.

- [LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 35–54. Springer, 2013.
- [LSS14] Adeline Langlois, Damien Stehlé, and Ron Steinfeld. Gghlite: More efficient multilinear maps from ideal lattices. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 239–256. Springer, 2014.
- [MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, pages 700–718, 2012.
- [MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007.
- [PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO (1)*, volume 8616 of *Lecture Notes in Computer Science*, pages 500–517. Springer, 2014.
- [PTT10] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal authenticated data structures with multilinear forms. In Marc Joye, Atsuko Miyaji, and Akira Otsuka, editors, *Pairing-Based Cryptography - Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 246–264. Springer Berlin Heidelberg, 2010.
- [Rot13] RonD. Rothblum. On the circular security of bit-encryption. In Amit Sahai, editor, *Theory of Cryptography*, volume 7785 of *Lecture Notes in Computer Science*, pages 579–598. Springer Berlin Heidelberg, 2013.
- [RS09] Markus Rckert and Dominique Schrder. Aggregate and verifiably encrypted signatures from multilinear maps without random oracles. In JongHyuk Park, Hsiao-Hwa Chen, Mohammed Atiquzzaman, Changhoon Lee, Tai-hoon Kim, and Sang-Soo Yeo, editors, *Advances in Information Security and Assurance*, volume 5576 of *Lecture Notes in Computer Science*, pages 750–759. Springer Berlin Heidelberg, 2009.
- [SOK00] Ryuichi Sakai, Kiyoshi Ohgishi, and Masao Kasahara. Cryptosystems based on pairing. In *SCIS 2000*, Okinawa, Japan, January 2000.
- [Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, 2005.
- [WC81] Mark N. Wegman and J.Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265 – 279, 1981.

A Parameter Selection

We now describe the parameter-generation procedure **PrmGen**, showing how to choose the parameters for our scheme. The procedure takes as input the security parameter λ , a DAG G with diameter d , and the class $\mathcal{C}$ of supported circuits. It outputs n, m, q and the Gaussian parameters s, σ . The constraints that these parameters need to satisfy are the following:

- It should be possible to efficiently sample from the input/error distribution $\chi = D_{\mathbb{Z},s}$, and the LWE problem with parameters n, m, q, χ should be hard. This means that we need (say) $s = \sqrt{n}$ and $q/s < 2^{n/\lambda}$.
- It should be possible to generate trapdoor for the $\mathbf{A}_v$'s, that enables sampling from **PreSample** with parameter σ . By Lemma 2.1, this means that we need $m = \Omega(n \log q)$ and $\sigma = \Omega(\sqrt{n \log q})$.
- For any supported circuit, the size of the error $\mathbf{E}$ at the output of the circuit must remain below $q^{3/4}$. Namely if the output is an encoding $\mathbf{D}$ of the plaintext matrix $\mathbf{S}$ relative to path $u \rightsquigarrow v$, then we need $\|[\mathbf{D}\mathbf{A}_u - \mathbf{A}_v\mathbf{S}]_q\| < q^{3/4}$.

Let us now analyze the error size in the system. We assume here that we use truncated Gaussian distributions, i.e. we condition $D_{\mathbb{Z},s}$ on the output being smaller than $b \stackrel{\text{def}}{=} s\sqrt{\lambda}$ (which only affect the distribution negligibly.) We similarly condition **PreSample** on the output being shorter than $B \stackrel{\text{def}}{=} \sigma\sqrt{\lambda}$. With our settings, we get $b \leq n$ and $B \leq n\sqrt{\log q}$. Hence the sample procedure always outputs $(\mathbf{S}, \mathbf{C}, \mathbf{D})$ with the plaintext satisfying $\|\mathbf{S}\| < b$ and the encoding matrices satisfying $\|\mathbf{C}\|, \|\mathbf{D}\| < B$.

To analyze the noise development, recall that when multiplying $\mathbf{A} \in \mathbb{Z}^{u \times v}$ by $\mathbf{B} \in \mathbb{Z}^{v \times w}$ we have $\|\mathbf{AB}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{B}\| \cdot v$. This means in particular that multiplying i encoding matrices we get an encoding matrix $\mathbf{D} \in \mathbb{Z}_q^{m \times m}$ with $\|\mathbf{D}\| < B^i m^{i-1}$ and similarly multiplying i plaintext matrices we get a plaintext matrix $\mathbf{S} \in \mathbb{Z}_q^{n \times n}$ with $\|\mathbf{S}\| < b^i n^{i-1}$. Regarding the error, one can show by induction that the product of i encoding matrices has an error $\mathbf{E} \in \mathbb{Z}_q^{m \times n}$ with

$$\|\mathbf{E}\| < b \cdot \sum_{j=0}^{i-1} B^j m^j b^{d-1-j} n^{d-1-j} < b \cdot i \cdot B^{i-1} m^{i-1}.$$

Given a class $\mathcal{C}$ of permitted circuits, we consider the canonical representation of the polynomials in this class as sums of monomials. Let D be a bound on the degree of these polynomials, R be a bound on the size of the coefficients, and N be a bound on the number of monomials. Note that in our setting, the degree-bound D cannot be larger than the diameter of the graph G (since G is acyclic and hence cannot have directed paths longer than d). The size of the error in this case could grow as much as $N \cdot R \cdot b \cdot D \cdot B^{D-1} m^{D-1}$. With $b \leq n$ and $B \leq n\sqrt{\log q}$, we thus have the constraint

$$\begin{aligned} q^{3/4} &> N \cdot R \cdot n \cdot D \cdot (n\sqrt{\log q})^{D-1} m^{D-1} \\ &= N \cdot R \cdot n^D \cdot m^{D-1} \cdot D \cdot (\log q)^{(D-1)/2}. \end{aligned} \tag{5}$$

Substituting $m = \Theta(n \log q)$, and $q = 2^{n/\lambda}$, we can use Eqn. (5) to solve for n in terms of λ, N, R and D . With $D \leq d$ and assuming the (typical) case of $R = \text{poly}(\lambda)$ and $N < d^d$, it can be verified

that this bound is satisfied using $n = \Theta(d\lambda \log(d\lambda))$. This setting yields $q = 2^{n/\lambda} = 2^{\Theta(d \log(d\lambda))} = (d\lambda)^{\Theta(d)}$ and $m = \Theta(n \log q) = \Theta(d^2 \lambda^2 \log^2(d\lambda))$.

Note that with this setting, each matrix $\mathbf{A}_v \in \mathbb{Z}_q^{m \times n}$ is of size $mn \log q = \Theta(d^4 \lambda^2 \log^4(d\lambda))$ bits. The public parameters typically contain just one or a handful of such matrices (corresponding to the source nodes in G), but the secret parameters contain all of them. Hence the secret parameters are of size $\Theta(|V| \times d^4 \lambda^2 \log^4(d\lambda)) = \Omega(d^5 \lambda^2 \log^4(d\lambda))$ bits. (We have $|V| > d$ since the diameter of G is d .) The encoding matrices are of dimension $m \times m$, but their entries are small, so they can be represented by roughly $m^2 \log n = \Theta(d^4 \lambda^2 \log^5(d\lambda))$ bits.

Working over a larger ring. As usual, we can get better parameters by working over larger rings, and let n denote the extension degree of the ring. In this case the matrices $\mathbf{A}$ are $m \times 1$ column vectors over the larger ring, and we can find trapdoors for these matrices already for $m = \Theta(\log q)$, and also the plaintext elements are now scalars (or constant-degree matrices).

This only affects Eqn. (5) or the solution $n = \Theta(d\lambda \log(d\lambda))$ by a constant factor, and hence shaves only a constant factor from the number of bits in $q = 2^{\Theta(d \log(d\lambda))}$, but now we have $m = \Theta(\log q) = \Theta(d \log(d\lambda))$. With each scalar in R_q represented by $n \log q$ bits, it takes $mn \log q = \Theta(d^3 \lambda \log^3(d\lambda))$ bits to represent each matrix $\mathbf{A}_v \in R_q^{m \times 1}$, and $\Theta(m^2 \log n) = \Theta(d^3 \lambda \log^4(d\lambda))$ bits to represent each encoding matrix with small entries.